The Thesis Committee for Akhil Kanu Shah
Certifies that this is the approved version of the following Thesis:

Sensor Tasking Utilizing Deep Reinforcement Learning in a Random Finite Set Framework

APPROVED BY

SUPERVISING COMMITTEE:

Brandon A. Jones, Supervisor

Maruthi R. Akella

Sensor Tasking Utilizing Deep Reinforcement Learning in a Random Finite Set Framework

by

Akhil Kanu Shah

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2018

Dedicated to Kanu, Chetna, Deesha, and Sapna.

Acknowledgments

There are countless people that deserve thanks for their aid in my journey. I would like to thank Dr. Jones and Dr. Akella for their help in preparing my thesis and for serving as my advisors during my time in graduate school. Also, special thanks for Dr. Jones for inviting me to join his research group, without which, I would not have been able to learn and explore many different topics.

I am also grateful to Alex Herz and Orbit Logic for funding this research and also their comments and help.

I would be amiss to not thank my colleague, Nicholas Ravago, for his help in navigating astrodynamics and advanced filters. Without his help, I would still be debugging to this day.

A special thanks goes out to Andrei, Arjun, Jhanani, and Rahul for letting me use them to bounce ideas and techniques to properly train neural networks. Their experience and advice was invaluable.

Finally, I would like to thank my family for their immense supprt throughout my education and letting me explore and navigate the world without complaint. Their guidance and love is what has allowed me to reach this state, and for that I am truly grateful.

Sensor Tasking Utilizing Deep Reinforcement Learning in a Random Finite Set Framework

Akhil Kanu Shah, M.S.E.
The University of Texas at Austin, 2018

Supervisor: Brandon A. Jones

There is a growing need to increase the capabilities of existing sensor arrays to monitor a large amount of space objects orbiting the Earth with a limited number of opportunities to observe these objects. Due to geopolitical considerations and financial cost, it is infeasible to create an array of sensors that can monitor each space object and accurately describe its state. Instead of brute force techniques by increasing the number of sensors worldwide, the current advancements in computational capability along with new algorithms for multi-target filtering and reinforcement learning has allowed a pathway to begin solving the non-myopic, heterogenous sensor tasking problem.

This work employs the labeled multi-Bernoulli filter in conjunction with advanced, deep reinforcement learning techniques such as the policy gradient Q-learning algorithm and deep Q-networks. The filter and reinforcement learning techniques are used together to track ten targets in geosynchronous orbit, while a linear Kalman filter and the reinforcement learning techniques are

used to evaluate their effectiveness in multi-agent learning scenarios. The future deployment of these algorithms and their specific logistical considerations are also discussed with potential solutions.

Table of Contents

Acknowledgments **v**

Abstract **vi**

List of Tables **xi**

List of Figures **xii**

Chapter 1. Introduction **1**
1.1 Motivation 2
1.2 Contribution 3

Chapter 2. Background **4**
2.1 Markov Decision Processes 5
2.2 Multi-target Filtering Techniques 7
2.2.1 Single Target Filtering 8
2.2.1.1 Square Root Unscented Kalman Filter 9
2.2.2 Multi-Target Filtering 11
2.2.2.1 Notation 12
2.2.2.2 Labeled Multi-Bernoulli Random Finite Sets . . 13
2.2.2.3 LMB Filter 14
2.3 Information Theoretic Reward 16
2.4 Reinforcement Learning Techniques 19
2.4.1 Policy Gradients 21
2.4.2 Value Function Methods 23
2.4.3 Recent Techniques 25
2.5 Multi-agent Learning 27
2.6 Artificial Neural Networks 28
2.6.1 Types of Neural Networks 29

2.6.2 Activation Functions 29
2.7 Sensor Tasking and Management Techniques 31

Chapter 3. Implementation and Simulations 34
3.1 Gaussian Mixture LMB Filter 36
3.2 Rényi and Cauchy-Schwarz Divergence 38
3.2.1 Rényi Divergence Implementation 38
3.2.2 Cauchy-Schwarz Divergence Implementation 40
3.3 Reinforcement Learning Implementation 41
3.3.1 Policy Gradient Q-Learning Implementation 41
3.3.2 Q-Learning Implementation 44
3.4 Neural Network Design 44
3.4.1 PGQL Neural Network Design 45
3.4.2 Q-Learning Network Design 46
3.5 Simulation 46
3.5.1 Linear 2-Dimensional Problem 48
3.5.2 Ten Target Geosynchronous Problem 50

Chapter 4. Results 52
4.1 Geosynchronous Orbit Tasking Results 52
4.1.1 Comparison of Reward Function 53
4.1.1.1 Rényi Divergence Results 53
4.1.1.2 Cauchy-Schwarz Divergence Results 60
4.1.2 Exploration Factor Effects on Training 65
4.1.3 Comparison with Off-Policy Methods 69
4.2 Multi-agent Scenario 70

Chapter 5. Conclusions 74
5.1 Observations 75

Chapter 6. Future Work 77
6.1 Convolutional Neural Networks 77
6.2 Missed Detections and Operation Considerations 79
6.3 Reinforcement Learning Fragility 80

Appendices **82**

Appendix A. Initial States for the Geosynchronous Satellites **83**

Bibliography **84**

List of Tables

2.1 Reinforcement learning algorithms' characteristics 21

3.1 Reinforcement learning and neural network tuning parameters for the space situational awareness case. 47

3.2 Reinforcement learning and neural network tuning parameters for the space situational awareness case. 47

3.3 Initial conditions for the 2-D problem. 50

4.1 Actions for the multi-action Q-learning scenario over time. . . 71

4.2 Actions for the multi-action PGQL scenario over time. 72

A.1 Initial Cartesian states in kilometers and kilometers per second, as appropriate. 83

List of Figures

2.1 Plots of the sigmoid, tanh, and ReLU activation function. . . . 30

3.1 Flow chart of the structure of the algorithms and their interactions when training occurs. 35
3.2 Flow chart of the structure of the algorithms and their interactions without training. 36
3.3 Initial starting conditions and sensor setup for the linear 2-D problem. 49

4.1 The Mahalanobis distance for the Rényi divergence for the full state over the entire night for all 10 targets. 54
4.2 The Mahalanobis distance for the Rényi divergence for just the position states over the entire night for all 10 targets. 55
4.3 The Mahalanobis distance for the Rényi divergence for just the velocity states over the entire night for all 10 targets. 55
4.4 Target 2 Errors with $\pm 3\sigma$ envelopes for the Rényi divergence. 57
4.5 Position OSPA error for the Rényi divergence. 59
4.6 Velocity OSPA error for the Rényi divergence. 59
4.7 The Mahalanobis distance for the Cauchy-Schwarz divergence for the full state over the entire night for all 10 targets. 60
4.8 The Mahalanobis distance for the Cauchy-Schwarz divergence for just the position states over the entire night for all 10 targets. 61
4.9 The Mahalanobis distance for the Cauchy-Schwarz divergence for just the velocity states over the entire night for all 10 targets. 62
4.10 Target 2 Errors with $\pm 3\sigma$ envelopes for the Cauchy-Schwarz divergence. 63
4.11 Position OSPA error for the Cauchy-Schwarz divergence. . . . 64
4.12 Velocity OSPA error for the Cauchy-Schwarz divergence. . . . 65
4.13 The Mahalanobis distance for the Cauchy-Schwarz divergence for the full state over the entire night for all 10 targets with $\lambda = 0.1$. 66

4.14 Target 1 Errors with $\pm 3\sigma$ envelopes for the Cauchy-Schwarz divergnce with $\lambda = 0.1$. 67
4.15 Position OSPA error for the Cauchy-Schwarz divergence with $\lambda = 0.1$. 68
4.16 Velocity OSPA error for the Cauchy-Schwarz divergence with $\lambda = 0.1$. 69
4.17 Mahalanobis distance of the full state for the Q-learning scenario. 70

Chapter 1

Introduction

As the space environment becomes crowded with more satellites and operators, it has become prudent to track these objects to ensure collisions are avoided and to keep an open and safe space environment. The Chinese ASAT test in 2007 showed the impact of a single collision in low-Earth orbit and the Iridium-Kosmos event in 2009 showed the need of more robust and accurate tracking methodologies [19, 20]. The current publicly available space catalog permits tracking the states of over 15,000 objects , but the uncertainty in this state information can cause operators to make ill informed decisions about their individual satellite.

The simplest method would be to increase the number of sensors available to track the current state catalog and any new objects that are launched into orbit. Unfortunately, this naive approach is unfeasible due to the financial burden of constructing these sensors and international borders. Instead, industry and academia has worked to develop more powerful algorithms for filtering and optimally choosing objects to observe.

1.1 Motivation

The goal of this work is to leverage these new advances to create a solution that can eventually minimize the number of sensors and the number of observations required to reasonably track a large number of space objects. More robust and accurate filtering techniques have been used in many different target tracking scenarios and are capable of generating more exact state knowledge. The state knowledge can then be used to solve the optimal sensor control problem via dynamic programming to find the optimal long-term decisions to maximize an information-theoretic reward. Unfortunately, due to the number of potential actions and the number of potential target states, dynamic programming is often computationally untractable. Instead, this work attempts to combine the multi-target filter with approximate methods, commonly referred to as reinforcement learning, as an attempt to solve the optimal control problem.

Even with these advancements, a single sensor would be unable to monitor and track all 15,000 objects and thus sensors coordinating and learning together may prove to be the best option. However, multi-agent learning proves to be a challenge to use with the current state of the art. For a heterogenous, non-myopic sensor tasking a solution a method that attempts to find the optimal, long-term reward is required, which is the main focus of joining multi-target filtering with reinforcement learning.

1.2 Contribution

This thesis utilizes the random finite set based labeled multi-Bernoulli filter as its multi-target filter and tests its compatibility with two reinforcement learning algorithms: policy-gradient Q-learning and Q-learning. The scenario constructed for this test utilizes a single sensor with ten targets in a geosynchronous obrit and each filter-learning pair shows promise. However, in terms of number of required observations to properly train, the former reinforcement learning algorithm outperforms the latter.

A simple two dimensional linear scenario was constructed to determine the feasibility of the reinforcement learning techniques with a linear Kalman filter. The results did not show a useable solution, but the tests show promise for future analysis and extensions. While it was not visted in this work, a goal for the future is to extend the ten taget case in a multi-agent environment with multiple sensors.

Finally, some avenues of exploration are discussed for the issues discovered during this work as an eventual means to deploy a fully realized filter, neural network, and training set for the space situational awareness problem.

Chapter 2

Background

The sensor tasking problem is the amalgamation of many different fields to solve both an optimization and a filtering problem. The optimization problem is formulated as a partially observable Markov decision process (POMDP), but utilizes the same ideas as Markov decision processes (MDPs). Due to the size of the state and action space, MDP solution techniques are intractable. Instead, the optimization problem is solved via approximate solutions formulated by neural networks and trained by reinforcement learning techniques. In this context, the state space can be described as the position and velocity of each target, and the action space as all of the potential sensing actions. An individual neural network with its supporting learning paradigm represents a single sensor.

Reinforcement learning requires full state knowledge at each decision point which is infeasible for a POMDP. Instead, Bayesian filtering techniques are utilized to construct an estimate of the full-state from the sensing actions derived from the neural network. This interaction between filter and network continues during the reinforcement learning process and, in the ideal case, the neural network converges to a solution, from which a policy can be generated

for further sensing actions. In this chapter, the mathematical framework and techniques from these different fields are described to develop a solution to the sensor tasking problem.

2.1 Markov Decision Processes

The sensor tasking or resource management problem is a subset of uncertain sequential decision-making problems, which utilize the Markov decision process (MDP) framework. Markov decision processes require the following information: the potential states (s), the potential actions (a), the transition probabilities as a function of the state and action ($P(s' \mid s, a)$), and a reward function as a function of the state and action $r(s, a)$. The goal then is to maximize the cumulative reward via the optimal policy or mathematically [49]:

$$J = \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)\right] \tag{2.1}$$

where π refers to a policy that contains all the actions a_t, and γ is a discount factor constrained by $0 < \gamma \leq 1$. It should be noted that the policy, π, is represented as the probability distribution of an action given the corresponding state that will maximize the reward. Eq. 2.1 is defined for a finite horizon problem, but the formulation is similar for an infinite horizon problem and is updated by replacing T with infinity. Eq. 2.1, however, is not trivial to solve and instead relies upon other solution techniques. The finite horizon policy case can be solved via backwards dynamic programming, which solves for the value of each state [49]. The infinite horizon problem utilizes two similar tech-

niques - policy iteration and value iteration. Policy iteration develops the optimal policy by assigning a value to each possible state and then determines the best possible action for each state. From there, new values are assigned, given the policy and a new policy is constructed until the policy converges. Value iteration takes a similar approach, but combines the two-step policy iteration into one step. At each iteration, only the values of the states are computed until the values converge, from which the optimal policy is computed by choosing the state with the highest value. Both policy and value iteration calculate the globally optimal policy. Each of these methods utilize the Bellman optimality equations [49]:

$$V_t(s_t) = \max_{a_t}(r(s_t, a_t) + \gamma \mathbb{E}(V_{t+1}(s_{t+1} \mid s_t))) \tag{2.2}$$

where V_t is the value function at some discrete time t, a_t is the action at time t, $r(s_t, a_t)$ is the reward at time t, s_t and s_{t+1} refer to the state at time t and $t+1$, respectively. For an MDP with known transition probabilities, the expectation term becomes $\sum_{s_{t+1}} P(s_{t+1} \mid s, a)V_{t+1}(s_{t+1})$, i.e., the sum of all the possible next states' probabilities scaled by the value at that state.

The MDP formulation has several issues which does not allow for a complete solution for the sensor tasking problem, but the core ideas can still be utilized. Value and policy iteration both provide a globally optimal solution, but each are computationally intractable with even a moderately sized action and state space. MDPs also require a model to characterize the $P(s' \mid s, a)$ and for most problems it is difficult to develop an accurate model. Since, the

transition probabilities are fundamental in producing an optimal policy, slight errors can potentially cause major deviations. For both the computational intractability and model-free solutions, techniques developed within the reinforcement learning or approximate dynamic programming communities can be employed [49, 4]. These techniques are derived as analogs to policy iteration and value iteration and are further discussed in Section 2.4.

The MDP constraint requires full state knowledge at each time instance to ensure that an appropriate action is credited with the corresponding reward. When full state information is unavailable, then the problem must be formulated as an MDP. However, it is possible to convert a POMDP into an MDP by utilizing the belief state, $\mathcal{I}$, which is composed of the previous observations and the actions [15]. Further, it is valid to describe $\mathcal{I}$ through the posterior distribution generated by such as Kalman filtering, or for the sensor tasking problem, multi-target filtering.

2.2 Multi-target Filtering Techniques

Multi-target filtering techniques attempt to utilize uncertain dynamics with noisy data sources to accurately determine the cardinality, or the number of targets, the target states, and a measure of accuracy. The measure of accuracy is typically given as the second statistical moment or the covariance. Multi-target filtering still utilizes single-target filtering algorithms and so, prior to any discussion on multi-target filtering techniques, it is prudent to discuss the single-target algorithms.

2.2.1 Single Target Filtering

Single-target filters require a model of the stochastic dynamics and measurements to determine an accurate representation of the true state. The dynamics are required to be Markovian [52], which allow the use of the Chapman-Kolmogorov equation:

$$p(\mathbf{x}_k \mid \mathbf{z}_{1:k-1}) = \int p(\mathbf{x}_k \mid \mathbf{x}_{k-1}) p(\mathbf{x}_{k-1} \mid \mathbf{z}_{1:k-1}) d\mathbf{x}_{k-1} \tag{2.3}$$

where $\mathbf{x}$ is the state, $\mathbf{z}$ is the observation, $p(\mathbf{x}_k \mid \mathbf{z}_{1:k-1})$ is the probability density of the current state conditional on the previous observations, $p(\mathbf{x}_k \mid \mathbf{x}_{k-1})$ is the probability density of the current state conditional on the previous state determined by the stochastic dynamics, and $p(\mathbf{x}_{k-1} \mid \mathbf{z}_{1:k-1})$ is the probability density of the previous state conditional on all previous observations. Eq. 2.3 is regarded as the prior distribution, which is then updated by Bayes theorem:

$$p(\mathbf{x}_k \mid \mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k \mid \mathbf{x}_k) p(\mathbf{x}_k \mid \mathbf{z}_{1:k-1})}{\int p(\mathbf{z}_k \mid \mathbf{x}_k) p(\mathbf{x}_k \mid \mathbf{z}_{1:k-1}) d\mathbf{x}_k} \tag{2.4}$$

where $p(\mathbf{z}_k \mid \mathbf{x}_k)$ is the probability density of the new measurement given the current state, and similarly, $p(\mathbf{z}_k \mid \mathbf{x}_{k-1})$ is the probability of the new measurement given the previous state. The other probabilities are determined from Eq. 2.3.

When the pdfs in Eqs. 2.3 and 2.4 are modeled as Gaussian pdfs, they are completely characterized by the means and covariance of $\mathbf{x}_k$, $\mathbf{x}_{k-1}$, and $\mathbf{z}_k$. Additionally, if the dynamics and observation models are linear, the Kalman filter can be utilized. For non-linear dynamics or observation models, the

extended Kalman filter, or the unscented transform can be utilized. The unscented Kalman filter is more accurate compared to the extended Kalman filter and is further discussed.

2.2.1.1 Square Root Unscented Kalman Filter

The unscented Kalman filter (UKF), also known as the sigma point filter, was first developed by Julier and Uhlmann to provide a better representation of Gaussian distributions undergoing non-linear transformations [56]. However, in this work, the focus is on the square-root unscented Kalman filter (SRUKF) and the proceeding discussions and equations will reflect that fact. The SRUKF, just as the UKF, begins with an *a priori* mean and covariance and a set of weights defined as:

$$W_0^m = \frac{\lambda}{n+\lambda} \tag{2.5}$$

$$W_0^c = \frac{\lambda}{n+\lambda} + (1-\alpha^2+\beta) \tag{2.6}$$

$$W_i^m = W_i^c = \frac{1}{2(n+\lambda)}, \quad i = 1, ..., 2n \tag{2.7}$$

where $\lambda = \alpha^2(n+\kappa) - n$, n refers to the dimension of the state, and α, β, and κ are all tuning parameters [6]. The superscript m or c refers to whether the weight is calculated for the mean or the covariance, respectively. Additionally, $2n+1$ sigma points are generated as:

$$\mathcal{X}_0 = \bar{\mathbf{x}} \tag{2.8}$$

$$\mathcal{X}_i = \bar{\mathbf{x}} + \sqrt{n+\lambda} L_i \tag{2.9}$$

$$\mathcal{X}_{n+1} = \bar{\mathbf{x}} - \sqrt{n+\lambda} L_i \tag{2.10}$$

where $\bar{\mathbf{x}}$ is the mean, L_i is the ith column of the lower Cholesky decomposition of the covariance. These sigma points are then passed through the non-linear dynamics and recovered by using the weights generated from Eqs. 2.5-2.7. The equations for recovering the updated mean and covariance are seen in Eqs. 2.11 to 2.13.

$$\bar{\mathbf{x}} = \sum_{i=0}^{2n} W_i^m \mathcal{X}_i \tag{2.11}$$

$$\bar{\mathbf{S}} = \text{qr}\left\{\left[\sqrt{W_1^{(c)}}(\mathcal{X}_{1:2n} - \bar{\mathbf{x}}) \quad \sqrt{\mathbf{Q}}\right]\right\} \tag{2.12}$$

$$\bar{\mathbf{S}} = \text{cholupdate}\left\{\left[\bar{\mathbf{S}} \quad \mathcal{X}_0 - \bar{\mathbf{x}} \quad W_0^{(c)}\right]\right\} \tag{2.13}$$

where qr is a function that performs a QR decomposition, and cholupdate is a function that adds a vector, $\boldsymbol{x}$, to a matrix, $\boldsymbol{A}$, without having to reconstruct the non-Cholesky form.

The measurement update portion of the UKF then requires the time-updated sigma points to pass through the observation model and returns the observation sigma points, $\mathcal{Z}$, and, similarly to Eqs. 2.11 and 2.13, the corresponding observation mean and covariance can be recovered. For completeness these equations are described in Eqs. 2.14 to 2.16.

$$\bar{\mathbf{z}} = \sum_{i=0}^{2n} W_i^m \mathcal{Z}_i \tag{2.14}$$

$$\bar{\mathbf{S}}_{zz} = \text{qr}\left\{\left[\sqrt{W_1^{(c)}}(\mathcal{Z}_{1:2n} - \bar{\mathbf{z}}) \quad \sqrt{\mathbf{R}}\right]\right\} \tag{2.15}$$

$$\bar{\mathbf{S}}_{zz} = \text{cholupdate}\left\{\left[\bar{\mathbf{S}}_{zz} \quad \mathcal{Z}_0 - \bar{\mathbf{z}} \quad W_0^{(c)}\right]\right\} \tag{2.16}$$

Before the measurement update equations are established, the cross-covariance

of $\bar{\mathbf{z}}$ and $\bar{\mathbf{x}}$ must be defined and can be found in Eq. 2.17.

$$\mathbf{S}_{xy} = \sum_{i=0}^{2n} W_i^{(c)} (\mathcal{X}_i - \bar{\mathbf{x}})(\mathcal{Z}_i - \bar{\mathbf{z}})^T \tag{2.17}$$

The measurement update equations are defined in Eqs. 2.18-2.21:

$$\mathbf{K} = (\mathbf{S}_{xz}\mathbf{S}_{zz}^{-T})\mathbf{S}_{zz}^{-1} \tag{2.18}$$

$$\mathbf{x} = \bar{\mathbf{x}} + \mathbf{K}(\mathbf{z} - \bar{\mathbf{z}}) \tag{2.19}$$

$$\mathbf{U} = \mathbf{K}\mathbf{S}_{xz} \tag{2.20}$$

$$\mathbf{S} = \text{cholupdate}\begin{bmatrix}\mathbf{S} & U & -1\end{bmatrix} \tag{2.21}$$

where $\mathbf{K}$ is the Kalman gain, $\mathbf{x}$ is the measurement updated mean state, $\mathbf{z}$ is the realized observation, and $\mathbf{P}$ is the measurement updated covariance. The single-target filtering allows for a discussion of multi-target filtering, and where single-target filtering plays an important role.

2.2.2 Multi-Target Filtering

Historically, three major frameworks exist for multi-target tracking - Multiple Hypothesis Tracking (MHT), Joint Probabilistic Data Association (JPDA), and Random Finite Sets (RFS) [50]. Subsequent discussions will focus on the RFS framework, however, it should be noted that MHT and JPDA are classical approaches to the multi-target filtering problem. A true Bayesian filter is intractable as it attempts to accurately describe the pdf via all of its moments. The RFS approach retains all of the moments of the state pdf with high computational effort. Hovever, other RFS approaches have been

developed to match the true state pdf up to the second moment to allow for tractability, which is described in further detail and used in this work.

Instead of operating solely on individual states, $\mathbf{x}$, and observations, $\mathbf{z}$, multi-target filtering operates on the space of every potential state, $\mathbb{X}$, and measurement, $\mathbb{Z}$. Similar to the discussion on single-target filtering, the starting point is the multi-target Chapman-Kolmogorov equation and the accompanying multi-target update from Bayes theorem, which are defined in Eqs. 2.22 and 2.23 [35]:

$$p(X_k|Z_{1:k-1}) = \int p(X_k|X_{k-1})p(X_{k-1}|Z_{1:k-1})\delta X_{k-1} \tag{2.22}$$

$$p(X_k|Z_{1:k}) = \frac{p(Z_k|X_k)p(X_k|Z_{1:k-1)}}{\int p(Z_k|X_k)p(X_k|Z_{1:k-1})\delta X_k} \tag{2.23}$$

where X and Z refer to the finite subsets of the state ($\mathbb{X}$) and measurement ($\mathbb{Z}$) spaces, respectively. Mathematically, X and Z are defined as:

$$X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \tag{2.24}$$

$$Z = \{\mathbf{z}_1, \ldots, \mathbf{z}_n\} \tag{2.25}$$

where each state, $\mathbf{x}_i$, and observation, $\mathbf{z}_i$, realization are required to be in the state, $\mathbb{X}$, and measurement, $\mathbb{Z}$, spaces.

2.2.2.1 Notation

Before further discussion of multi-target filters can occur, some notation is introduced to simplify the discussion of the filter. First, the multi-object

exponential is defined in Eq. 2.26.

$$f^X = \prod_{x \in X} f(x) \tag{2.26}$$

Additionally, the Kronecker delta defined as

$$\delta_A(B) = \begin{cases} 1, & \text{if } B = A \\ 0, & \text{otherwise} \end{cases} \tag{2.27}$$

and finally, the inclusion function defined as

$$1_A(B) = \begin{cases} 1, & \text{if } B \subseteq A \\ 0, & \text{otherwise.} \end{cases} \tag{2.28}$$

For the purposes of the multi-target filter section, lower-case letters refer to single-target realizations, upper-case letters refer to sets, and blackboard letters refer to spaces.

2.2.2.2 Labeled Multi-Bernoulli Random Finite Sets

A Bernoulli RFS, as its name implies, employs the use of a Bernoulli distribution for its cardinality. This probability distribution of the RFS is defined as [35]:

$$\pi(X) = \begin{cases} 1 - r, & X = \emptyset \\ r \cdot p(x), & X = \{x\} \end{cases} \tag{2.29}$$

where r refers to the probability of existence, and $p(x)$ denotes the pdf of the state $x \in \mathbb{X}$. The multi-Bernoulli RFS can be constructed from the union of M independent Bernoulli RFSs and is defined by each individual parameter r and pdf p. The pdf of the multi-Bernoulli RFS is defined as [35]:

$$\pi(\{x_1, ..., x_n\}) = \prod_{j=1}^{M} (1 - r^{(j)}) \sum_{1 \leq i_1 \neq ... \neq i_n \leq M} \prod_{l=1}^{n} \frac{r^{(i_l)} p^{(i_l)}(x^{(l)})}{1 - r^{(i_l)}} \tag{2.30}$$

The density can be interpreted as summing over the probabilities of all combinations of the n objects within the set X, which is then multiplied with the probability that the object exists or not over all of the objects.

Vo and Vo introduced labels, $\ell \in \mathbb{L}$, as a means to estimate the identity of a particular target [62]. Labels must be unique for each possible target and so $|X| = |\mathcal{L}(X)|$, where $\mathcal{L}$ is the projection of the set of single-target realizations onto the label space $\mathbb{L}$. For ease of notation, the distinct label indicator is introduced and defined as:

$$\Delta(X) = \delta_{|X|}(|\mathcal{L}(X)|) \tag{2.31}$$

It is then possible to construct the labeled multi-Bernoulli RFS density parameterized by $\{(r_\ell, p_\ell)\}_{\ell \in \mathbb{L}}$ as:

$$\pi(X) = \Delta(X) w(\mathcal{L}(X)) p(x, \ell)^X \tag{2.32}$$

where

$$w(L) = \prod_{i \in \mathbb{L}} (1 - r^{(i)}) \prod_{\ell \in L} \frac{1_{\mathbb{L}}(\ell) r^{(\ell)}}{1 - r^{(\ell)}} \tag{2.33}$$

2.2.2.3 LMB Filter

The LMB filter is an approximation of the δ-GLMB filter, which is further discussed in [62]. The LMB filter is closed under the prediction step because it follows the δ-GLMB filter prediction step as a one term δ-GLMB

[50]. The prediction equations are then

$$r_{S,k|k-1}^{(\ell)} = \eta_S(\ell) r_{k-1}^{(\ell)} \tag{2.34}$$

$$p_{S,k|k-1}^{(\ell)} = \frac{\int p_s(x_{k-1},\ell) f(x|x_{k-1},\ell) p_{k-1}(x_{k-1},\ell) dx_{k-1}}{\eta(\ell)} \tag{2.35}$$

$$\eta(\ell) = \int p_s(x_{k-1},\ell) p_{k-1}(x_{k-1},\ell) dx_{k-1} \tag{2.36}$$

The update step of the LMB filter occurs on the δ-GLMB, but prior to the next prediction step, the δ-GLMB is approximated to an LMB. Truncations to the full hypothesis set are performed for the purpose of tractability before the δ-GLMB is updated by the measurements. The first truncation occurs during the gating process, where observations are tested to see if they are near an object. The second truncation is performed by employing an algorithm to choose a subset of the hypothesis after gating is performed. The hypothesis set is generated from the δ-GLMB, where each potential measurement to state association is considered. The second truncation is done by choosing the N most significant hypotheses by employing Murty's algorithm [39]. Further discussion of Murty's algorithm and its implementation can be found in [39]. After both hypothesis truncations are performed, the update equations defined in Eq. 2.37-2.42 are used.

$$r_{k|k}^{(\ell)} = \sum_{(I,\theta)\in\mathcal{F}(\mathbb{L})\times\Theta} w^{(I,\theta)}(Z) 1_I(\ell) \tag{2.37}$$

$$p_{k|k}^{\ell}(x) = \frac{1}{r^{(\ell)}} \sum_{(I,\theta)\in\mathcal{F}(\mathbb{L})\times\Theta} 1_I(\ell) w^{(I,\theta)}(Z) p^{(\theta)}(x,\ell) \tag{2.38}$$

where

$$w^{(I,\theta)}(Z) \propto w_{k|k-1}^{I} \left[n_Z^{(\theta)} \right]^I \tag{2.39}$$

$$p^{(\theta)}(x,\ell) = \frac{p_{k|k-1}(x,\ell)\psi_Z(x,\ell;\theta)}{\eta_Z^{(\theta)}(\ell)} \tag{2.40}$$

$$\eta_Z^{(\theta)}(\ell) = \int p_{k|k-1}(x)\psi_Z(x,\ell;\theta)dx \tag{2.41}$$

$$\psi_Z(x,\ell;\theta) = \begin{cases} \frac{p_D(x,\ell)g(z_{\theta(\ell)}|x,\ell)}{\kappa(z_{\theta(\ell)})}, & \text{if } \theta(\ell) > 0 \\ 1 - p_D(x,\ell), & \text{if } \theta(\ell) = 0 \end{cases} \tag{2.42}$$

$p_D(x,\ell)$ is the single-target probability of detection, $\kappa(\cdot)$ is the clutter intensity for a specific measurement.

2.3 Information Theoretic Reward

The reward function for many sensor management problems is defined as the information gained due to an update of the state via measurements. Specifically, the goal is to choose the measurements that maximizes the information gained, which is computed by comparing *a priori* and *a posteriori* pdfs. There are a variety of choices to quantify the information gained.

The earliest method to quantify information was developed by Shannon and is defined in Eq. 2.43,

$$H = -\sum p(x) \log p(x) \tag{2.43}$$

where x is a discrete random variable, and p is the probability mass function (pmf) [55]. The Shannon entropy can provide the information gained by subtracting the information of a random variable from the conditional entropy

of x given an observation y, i.e., the difference between the *a priori* and *a posteriori* pdfs.

Another choice is the Fisher information matrix which is a metric utilized internally in Kalman filter updates. The Fisher information matrix is defined in Eq. 2.44,

$$F = H^T R^{-1} H^T \tag{2.44}$$

where H is the observation matrix in a linear observation model or the Jacobian of a non-linear observation model, and R is the sensor error covariance. The reward function for an MDP formulation requires a scalar-valued function, and so the Fisher information matrix must be reduced to a scalar form by taking the trace or the determinant. The Fisher information matrix has limited use due to its reliance on a Gaussian model and continuously differentiable model [36].

An alternative, defined by Williams, et al. [66], utilizes Lyapunov exponents from Lyapunov stability analysis as a measure of divergence. Lyapunov exponents are approximated, in general, by the normed difference between points along the trajectory multiplied by an exponential, i.e.:

$$||\mathbf{x}_t - \mathbf{x}_{t-1}||e^{\lambda \Delta t} \tag{2.45}$$

where, $\mathbf{x}$ is the state along a trajectory, t is the time, Δt is the difference in time, and λ is the Lyapunov exponent. This approximation is only valid for when the normed difference is much less than unity [66]. For the sensor management problem, the largest Lyapunov exponent is used and is calculated

using the covariance of the state which can be seen in Eq. 2.46:

$$\lambda_k = \frac{1}{t_k} \log_2 \frac{\sqrt{\mathrm{Tr}(\mathbf{P}_k)}}{\sqrt{\mathrm{Tr}(\mathbf{P}_0)}} \tag{2.46}$$

where $\mathbf{P}_k$ is the covariance of the target at the current time, $\mathbf{P}_0$ is the initial covariance, and t_k is the current time.

A separate class of divergence metrics exist to fully quantify the difference between two pdfs. The Kullback-Leibler (K-L) divergence is akin to the Shannon Entropy defined in Eq. 2.43. The K-L divergence is defined in Eq. 2.47 [15]:

$$D_{KL} = \int p(x) \log \frac{p(x)}{q(x)} dx \tag{2.47}$$

where $p(x)$ and $q(x)$ are the two pdfs that are being compared. The K-L divergence can be considered as a subset of the Reńyi α-divergence. The Reńyi α-divergence is defined in Eq. 2.48 [15]:

$$D_\alpha = \frac{1}{\alpha - 1} \log \int p(x)^\alpha q(x)^{1-\alpha} dx \tag{2.48}$$

where $p(x)$ and $q(x)$ follow from the previous definition, and α scales the importance of each distribution and is constrained by $0 < \alpha < 1$. In the special case, where α approaches one, the Reńyi α-divergence becomes the K-L divergence. Hero, et al. [14], by performing a search has shown that an α of 0.5 computes the largest discrimination between the two densities. Unfortunately, the Reńyi α-divergence does not produce a closed form solution for multi-target pdfs produced by the LMB filter and instead must be numerically computed [51].

Instead, for the multi-target filters mentioned previously, an analytical form of the Cauchy-Schwarz (C-S) divergence does exist and have been characterized for the SSA problem [2, 5]. The C-S divergence is then calculated as seen in Eq. 2.49 [2].

$$D_{CS} = -\ln \frac{\langle f, g \rangle_\mu}{\|f\|_\mu \|g\|_\mu} \tag{2.49}$$

where $\langle f, g \rangle_\mu = \int f(X)g(X)\mu dX$ and $\|f\|_\mu = \sqrt{\langle f, f \rangle_\mu}$. f and g are probability density functions and μ is a reference measure. Even though the C-S divergence allows for an analytical solution, Gehly, et al. [9], shows that its use in sensor tasking cases is limited when utilizing Gaussian mixtures. The C-S divergence is scaled only by the individual covariance matrices and so a potential target with a smaller initial uncertainty will continually be tasked.

2.4 Reinforcement Learning Techniques

As described in the MDP formulation section in Section 2.1, as the number of potential states and actions increase, it is computationally intractable to utilize any of the solution methodologies described for MDPs. Instead, to overcome the computational intractability, function parameterizations are utilized to quickly generate actions; however, these parameterizations are required to be trained by interacting with the environment, which can take time. The reward function is the main interaction with the environment and its mimicry and encapsulation of the environment is important for successful training of the function parameterization.

Reinforcement learning is akin to psychological conditioning where, for

a certain action, a reward is given to reinforce or punish the particular behavior. Similarly, the goal of reinforcement learning is to train some function parameterization, e.g, a neural network, based on the reward from it immediate action and how that action may affect future rewards. The training achieves a near-optimal solution within an infinite horizon as the function parameterization interacts more with the environment.

Reinforcement learning provides mathematical paradigms to quickly and efficiently train simple function parameterizations, but in recent years, with advancements in computational power, reinforcement learning has become capable of handing complex, non-linear environments as well. Policy gradient and Q-learning are the two major methodologies in which reinforcement learning techniques fall into, but the community is now attempting to merge the two so that each categories' benefits create a more robust, flexible algorithm. Before discussing the specifics of each technique, Table 2.1 summarizes the algorithms and their specific characteristics.

Table 2.1: Reinforcement learning algorithms' characteristics

Algorithm	Off/On-Policy	Policy/Value Iteration Based	# of Neural Networks	Year Developed
REINFORCE Policy Gradient	On	Policy	1	1992 [67]
TD-Learning	Off	Value	1	1988 [58]
Q-Learning	Off	Value	1	1992 [65]
Actor-Critic	On	Policy	2	1999 [59]
Deep Q-Networks	Off	Value	2	2013 [38]
A3C	On	Policy	2	2016 [37]
PGQL	On	Both	4	2016 [45]
PCL	Off	Both	2	2017 [40]
Soft Actor-Critic	Off	Both	4	2018 [11]

2.4.1 Policy Gradients

Policy gradients attempt to solve a similar objective function to the one seen in Eq. 2.1, but the policy is constructed by some function parameterization, typically a neural network. The modified objective function is then

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \mid \theta\right] \tag{2.50}$$

where θ is the parameterization of the policy. Policy gradients begin by initializing the parameterization to some random values and while interacting with the environment iteratively update the policy based on the reward function and the gradients generated from Eq. 2.50. The gradients are calculated by directly differentiating the objective function from Eq. 2.50 and applying the "Reward Increment = Non-negative Factor times Offset Reinforcment times

Characteristic Eligibility" (REINFORCE) technique [67]. The gradients for the policy gradient algorithm are calculated via

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\left(\sum_{t\in T} \nabla_\theta \log \pi_\theta(a_t|s_t)\right)\left(\sum_{t\in T} r(s_t|a_t)\right)\right] \tag{2.51}$$

Policy gradients generally converge faster to a result than Q-learning techniques and are less susceptible to pitfalls found in complex environments since it iteratively finds the best policy in some policy space. However, policy gradients tend to converge onto local minima and are less prone to explore other options in the policy space. Policy gradient techniques are less robust to the high variance and bias and can skew the policy into a local minima.

In order to overcome the variance, Williams introduced a baseline $b \in \mathbb{R}$ to subtract from the reward in Eq. 2.51, i.e.:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\left(\sum_{t\in T} \nabla_\theta \log \pi_\theta(a_t|s_t)\right)\left(\sum_{t\in T} r(s_t|a_t) - b\right)\right] \tag{2.52}$$

It is possible to find b such that the variance of the gradients are minimized [48]. However, it is difficult to find a single parameter for all possible trajectories as the optimal b potentially has a spatial and/or a temporal dependence [48]. Instead, b can represented as a function parameterized by w, $\hat{V}_\pi(s|w)$, that learns the current value of the state. Training $\hat{V}_\pi(s|w)$ can be accomplished using temporal difference learning, which is described at greater length in Sec. 2.4.2. Methods that involve a function parameterization for the baseline instead of a real number are referred to as actor-critic methods. Actor-critic methods with a perfect critic will optimally reduce the variance of the gradients, but gener-

ally the critic is not perfect while it is being trained and therefore introduces a bias.

2.4.2 Value Function Methods

Temporal difference learning was introduced by Sutton as a model-free predictive method by assigning credit to specific actions that generate specific rewards, whereas earlier methods relied on the difference between the predicted outcome and the actual outcome [58]. The temporal difference algorithm was not explicitly developed for function parameterizations, but have been extended to them with success. The method, in the function parameterization case, is done by taking the gradient of the loss function described as

$$J(w) = \mathbb{E}\left[(V(s_t|w) - (r(s_t, a) + \gamma V(s_{t+1}|w))\right]^2 \tag{2.53}$$

where w is the parameterizing variable [49]. However, the loss function in Eq. 2.53 has a dependency on w in both the current value, $V(s_t \mid w)$ and the next value, $V(s_{t+1} \mid w)$, and when updating the w via a gradient method, the target is updated as well [38]. This loss function then causes instability during the training process and the value function may not be accurately characterized. Instead, the parameters w for the target network are kept fixed for some epoch and periodically updated.

Q-learning is a subset of temporal difference learning which operates on the state-action value function rather than the value function and was developed by Watkins [65]. The state-action value, otherwise known as the

Q-function, is related to the value function by

$$V(s) = \max_{a} Q(s, a). \tag{2.54}$$

The Q-function provides a simple method to determine the optimal policy. The policy is found by taking the argmax of the Q-function. A policy can be generated from the value function, Eq. 2.2, but it is not as trivial to compute. Utilizing the argmax, of the Q-function does not allow for exploration of the environment. Instead, the action is generally selected through an ϵ-greedy method, where the best action is chosen with probability $1 - \epsilon$, and a random action is chosen with probability ϵ. Of course, ϵ must be less than one. Then, the loss function becomes

$$J(w) = \mathbb{E}\left[(Q(s_t, a|w) - (r(s_t, a) + \gamma \max_{a'} Q(s_{t+1}, a'|w^{-}))\right]^2 \tag{2.55}$$

where a' is the action taken at the next state s_{t+1}, w^{-} is some previous parameterization of the Q-function for the target that is updated to w with some periodicity [64].

Both temporal difference learning and Q-learning require uncorrelated, non-sequential samples to minimize training time. Lin introduced the idea of experience replay, also known as replay buffers, to increase the stability and reduce both training time and number of samples to obtain an optimal Q-function [31]. Experience replay stores past interactions with the environment in a tuple of $\{s, a, r, s'\}$, where s' is the next state after taking action a, and when training a random subset of the buffer is chosen to train the Q-function

parameterization. Due to the nature of experience replay, any action from any policy can be chosen to update the Q-function. Schaul, et al., [53] introduced prioritized replay to choose prioritized tuples with the highest difference in the expected value and the returned value. Prioritized replay has shown improvement in both training time and the quality of the resulting policy.

Q-learning arrives at a better solution at the cost of training time and generally requires GPUs to perform training, whereas policy gradients can be trained on CPUs. Q-learning is not readily applicable to continuous action-spaces, however, strides have been made by Lillicrap, et al. [30] to incorporate them. The machine learning community has also been focused on advancing Q-learning to other avenues such as multi-agent learning as well.

2.4.3 Recent Techniques

Due to recent developments in computational capability, complex and non-linear environments can be trained with great success. Mnih, et al. [38], developed deep Q-networks (DQN) and used Q-learning with experience replay on the Atari 2600 suite of games utilizing frames from the game showed the trained network to vastly outperform humans. Van Hasselt, et al., [61] introduced Double Deep Q-networks (DDQN) to mitigate the overestimation and overoptimism from a single DQN and shows greater overall reward when compared to DQN. DQN has also been extended to POMDPs by leveraging recurrent neural networks (RNNs) on the same Atari 2600 suite, but with the addition of flickering and also has been shown to vastly outperform humans in

first person shooters [12, 24].

Mnih, et al. [37], also introduced asynchronous advantage actor-critic (A3C) where many worker agents are instantiated and perform tasks on identical environments. After an episode is completed or some time has passed, the gradients are passed to a global agent, which accumulates them, and then copies this new network to each worker. A3C generates many independent samples and for the same tasks is able to outperform DQN and train significantly faster [37].

Finally, there has been work to combine actor-critic methods and deep Q-learning. Lillicrap, et al. [30], utilized replay buffers to update the actor and critic networks to create the deep deterministic policy gradient (DDPG) alogrithm. Additionally, O'Donaghue, et al. [45], showed the equivalence of a modified Q-learning technique with actor-critic techniques and developed the policy gradient Q-learning (PGQL) algorithm similar to DDPG and showed that the algorithm outperformed both DQN and A3C on the Atari 2600 in most scenarios. Similarly, path consistency learning was developed by Nachum, et al. [40], which attempts to utilize only one network for both state and action values in an off-policy manner. Haarnoja, et al. [11], developed an algorithm that attempts to be immune to learning rates, structure of the neural networks, and the shape of the reward as DDPG, PGQL, and PCL can fail due to these parameters.

2.5 Multi-agent Learning

Advances in reinforcement learning over the years have focused on single agent interaction with an environment and have not extended to multi-agent interactions. In the context of the sensor tasking problem, multi-agent learning is important as each reinforcement learning agent represents a singular sensor or a group of multiple similar sensors. Multi-agent learning, however, breaks the Markovian assumption that the current state holds the information to choose the next optimal state and any notion of convergence or feasibility from reinforcement learning does not necessarily hold [47]. It can also cause instability in learning as a single agent is not learning a stationary environment, but an evolving one, where other agents interactions change the environment. The optimal policy for a single agent is then dependent on the policies from other agents, which can cause instability.

Multi-agent learning can be done via team-learning where a single, global network controls all of the agents. This technique faces the same combinatorial explosion of states and actions that prevents value iteration and policy iteration from being tractable solutions to the individual agent learning problem. A tractable option is concurrent learning where each agent's goal is to modify its own behavior to collect the highest reward. Concurrent learning works best when the agents are disjoint and largely independent [46].

The homogeneity of the agents allows for a smaller search space to find the optimal policy at the cost of specialization. Heterogeneous multi-agent learning generally provides better results than homogeneous agents due

to specialization at the expense of tractability. Specialization allows for individual agents to focus on specific goals, allowing the other agents to pursue their own agendas. Specialization is predominantly controlled by the credit each agent is assigned after an action is performed [46]. A global reward encourages cooperation between the agents, but cannot be efficiently computed, and does not reward beneficial agents nor punish lazy agents. Local rewards encourage individual agents to act selfishly and lowers specialization, but is computed efficiently and does reward beneficial agents.

Most concurrent learning methods rely on stochastic game theory, specifically Nash equilibria to perform multi-agent learning [46]. However, DQN has been utilized without minimal modification and shown promising results. Foerster, et al. [8], has shown that for cooperative multi-agent learning in real-time strategy video games has shown promise with unmodified DQN. Further, it was shown that DQN can be augmented with fingerprinting where each agent predicts the action of the other agents via Bayesian inference and/or with importance sampling where the loss function is augmented with the known policies of the other agents at any time [8]. Both of these methods, in combination or individually, outperform DQN and vanilla Q-learning.

2.6 Artificial Neural Networks

Artificial neural networks (ANNs) have a rich history in machine learning and have become the most commonly used function parameterization. ANNs generate an output from known inputs by activating the appropriate

neurons. The error between the expected outputs or the reward function is then used to train the network to provide the best solution. There are three major types of ANNs - feed-forward networks, convolutional networks (CNNs), and recurrent networks (RNNs). A fourth type, generative adversial networks (GANs), was introduced in 2014 and are an attempt to more closely mimic a biological network [10].

2.6.1 Types of Neural Networks

Feed-forward networks are used for the classification problem and can handle any type of input, however, struggle when the number of inputs are large and require many hidden layers. CNNs are primarily used for image classification since they reduce the original input size in subsequent convolutions and poolings. The convolutions and poolings are used to extract important features from an image, such as edges, to detect and classify the object in the image [25]. RNNs are utilized when some portion of previous outputs are required to form the current output. RNNs are used widely in the natural language processing (NLP) field to predict words in sentences and paragraphs. Further discussions on ANNs will focus on feed-forward networks, but most of the information present can be extended to both CNNs and RNNs.

2.6.2 Activation Functions

Inside each individual neuron, each input is multiplied by a weight and summed with a bias and then passed through an activation function. Different

activation functions exists and more recent developments have focused on ensuring that the gradients for each bias and weight are smooth and provide an accurate measure of difference from incorrect outputs or high rewards. Three activations functions, sigmoids, hyperbolic tangent, and rectified linear units (ReLU), are primarily used within the machine learning community. Sigmoid neurons approximate the step function and allow for smooth gradients. The sigmoid function is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.56}$$

where z is the input into the neuron, and can be seen graphically, along with the other activation functions, in Fig. 2.1 [43].

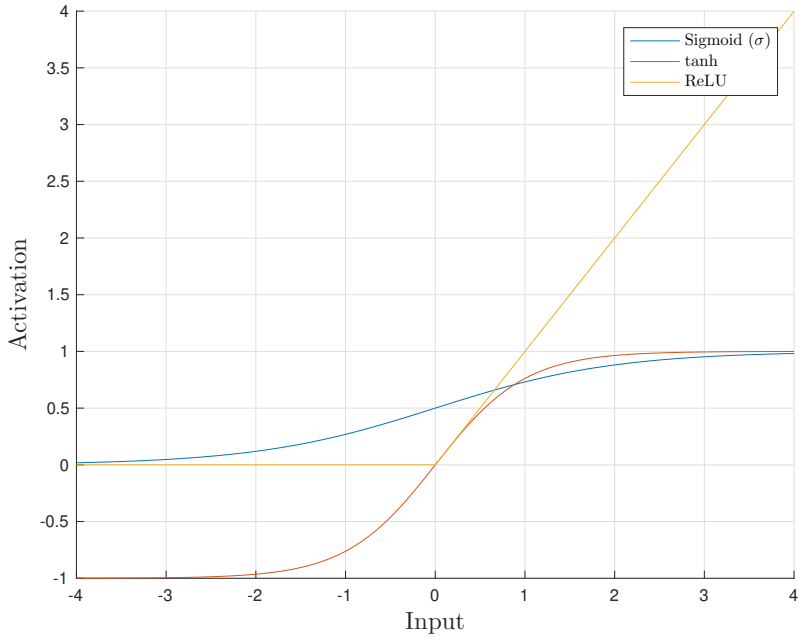

Figure 2.1: Plots of the sigmoid, tanh, and ReLU activation function.

For faster convergence, the mean of the input variable into a neuron should be near zero with standard deviation one [27]. The general sigmoid function defined in Fig. 2.1 is above the origin and so on average the inputs are not near zero for the other hidden layers. Instead, sigmoids that are normalized to the origin, such as the tanh function, are utilized as an alternative. The tanh function is sigmoid-like, i.e., $\tanh(x) = 2\sigma(2x) - 1$, and both are susceptible to vanishing gradients near maximum and minimum activations which slows down the learning process.

Instead, Nair and Hinton developed the use of ReLU activation functions to mitigate the vanishing gradient problem [41]. The ReLU activation is defined as $\max(0, z)$ and has a non-negative activation output. The output may violate the zero-mean condition, but empirically ReLU activations have faster convergence than other activation functions when used in deep networks [26]. Additionally, the ReLU activation is easier to calculate, its gradients are linear except at $z = 0$, and it does not fall prey to the vanishing gradients problem. However, unlike sigmoid and sigmoid-like functions, the activations are unbounded, which can cause issues in both gradient calculations and the output. Regardless of these downsides, ReLU activation functions are the preferred activation function by the machine learning community [26].

2.7 Sensor Tasking and Management Techniques

Predominantly, sensor management techniques have been applied to terrestrial applications such as airborne platforms monitoring ground- or sea-

based targets [22, 44]. Most techniques forgo optimality over a timeframe to improve computational tractability by utilizing a 'greedy' or a myopic solution where the objective is to maximize the reward function given the immediate state. Myopic strategies can be found in [22, 44, 34, 23, 66]. Myopic strategies range from solving a linear program, as in the case of Spencer, et al. [66], or by employing multi-armed bandits. Multi-armed bandits require careful structuring and a model and thus are limited in application. However, Wang has shown that myopic policy generated via multi-armed bandits are optimal [63].

Non-myopic solutions choose actions that maximize the reward over an entire trajectory in either a finite-horizon or an infinite-horizon time space. Non-myopic solutions are generally prescribed by normal roll-out methods [3, 22] or completely-observable roll-out [2, 9, 33, 29]. Sunberg, et al. [57], utilized receding horizon open-loop control to mitigate the combinatorial explosion on a simple sensor management problem. Reinforcement learning techniques have also been applied to sensor tasking problem. Q-learning has been applied by Kreucher, et al. [21], and vanilla policy gradient and A3C methods by Linares, et al. [32, 33].

Different types of reward functions have also been utilized for the sensor management problem. The majority of sensor management problems have used some form of a divergence metric. The K-L divergence has been used in [18], and Hero developed the use of the Reńyi divergence which is used in [9, 22, 21]. The LMB filters have closed form solutions of the C-S divergence and have been utilized in [9, 2].

Most often single-target filters such as the EKF or UKF are used as in [66, 7, 33, 32], but with the advent of multi-target filters, JPDA filters have been utilized in [22, 21, 29]. Gehly, et al. [9], and Beard, et al. [2], has utilized the PHD, CPHD, and LMB filter in their sensor tasking applications.

However, none of these works truly combine the use of multi-target with reinforcement learning to create a quick, deployable non-myopic solution. The non-myopic solutions that utilize roll-outs require long lead times to generate new schedules, whereas reinforcement learning algorithms can run in both an on-line or off-line capability. The downside is the long training time required before a sensor trained with reinforcement learning is capable of providing similar results to the roll-out algorithms. The closest analog to the work presented here is the combination of the JPDA filters and Q-learning with relatively small neural networks [21].

Chapter 3

Implementation and Simulations

This chapter will focus on the design considerations and implementation of the specific techniques discussed in Chapter 2. The LMB filter and the reward function chosen (Rényi divergence and Cauchy-Schwarz divergence) are agnostic to the type of pdf representation, but for ease of computation a Gaussian mixture approximation is used. Additionally, the neural network design and the implementation of the PGQL and the Q-learning algorithms are detailed.

While each function in the sensor tasking problem seems to be disjoint, there is a flow and deep interactions between each componeent. The LMB filter serves to reformulate the POMDP into an MDP and the states are fed into the neural networks to derive a sensing action through an actor network or a Q-network. The sensing action is then passed onto the sensor to generate observations for the LMB filter, which, in turn, updates the states with the new information. The divergence in the state pdfs as the reward function is then passed onto the training algorithms along with the state, the updated state, the action, and the reward. This cycle continues until training is deemed complete. For a non-training scenario, the reward function calculation and

training is not utilized. Each scenario is also detailed pictorally in Fig. 3.1 and 3.2.

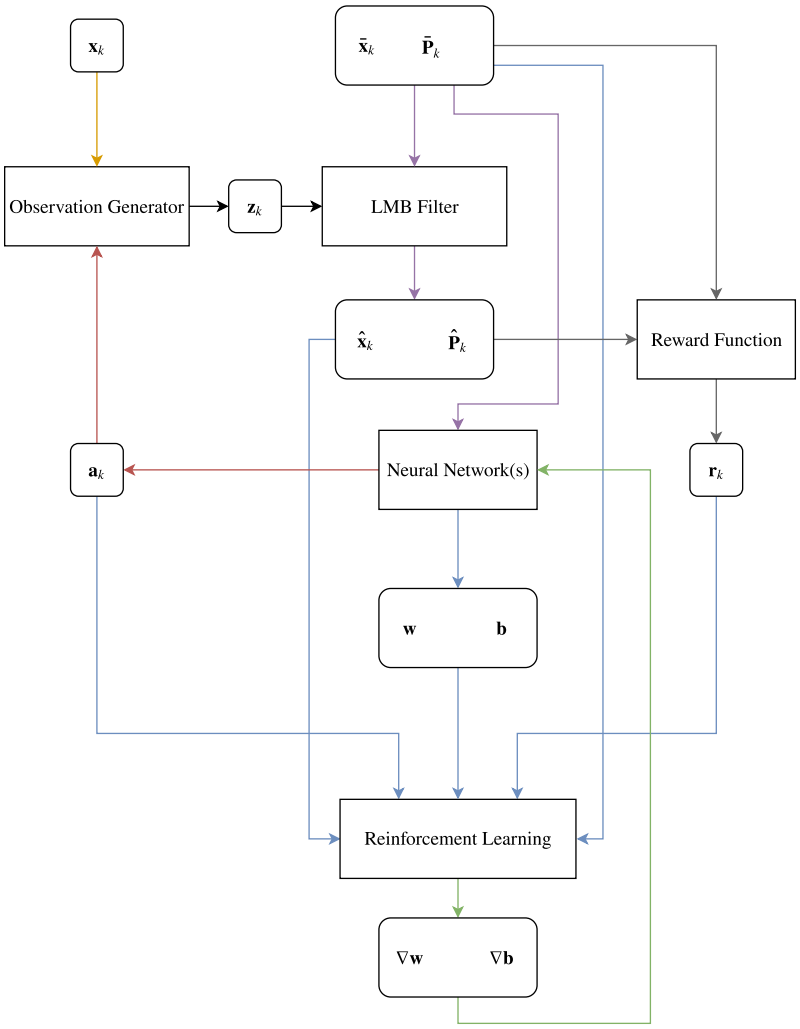

Figure 3.1: Flow chart of the structure of the algorithms and their interactions when training occurs.

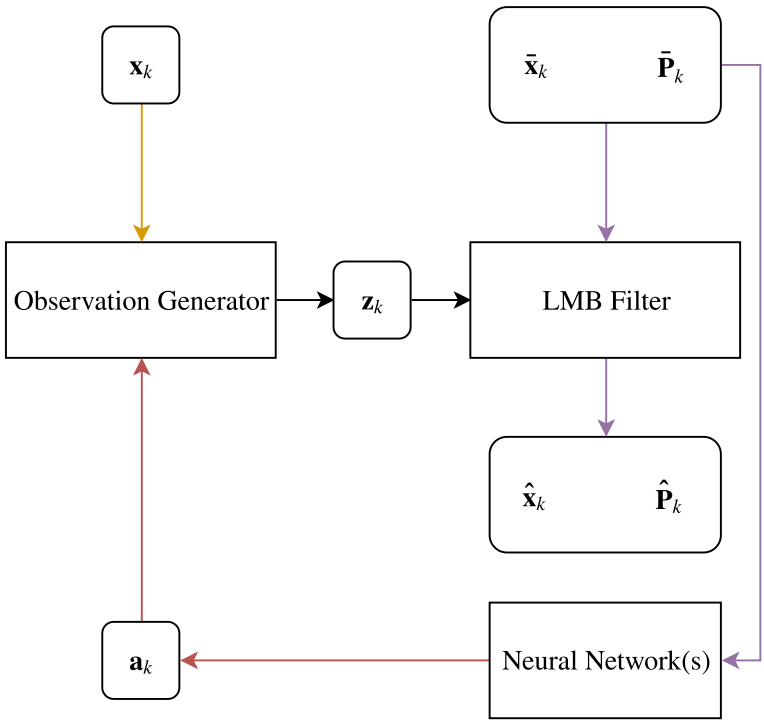

Figure 3.2: Flow chart of the structure of the algorithms and their interactions without training.

3.1 Gaussian Mixture LMB Filter

The LMB filter can work with any target-level pdf definition $p(x)$ in Eq. 2.29 and in this work it is represented as the sum of weighted Gaussian mixtures (GMMs). The pdf, $p(x)$, then can be described as:

$$p(x) = \sum_{j=1}^{J} \omega^{(j)} \mathcal{N}\left(x; \bar{x}^{(j)}, P^{(j)}\right) \tag{3.1}$$

where ω refers to the weight of a Gaussian component, and $\sum_{j=1}^{J} \omega^{(j)} > 0$, and $\bar{x}^{(j)}$ and $P^{(j)}$ refers to the propagated mean and covariance of the component, respectively. With the assumption on the pdf, the prediction equations, Eqs. 2.34 and 2.35, become:

$$r_{S,k|k-1}^{(\ell)} = p_s(\ell) r_{k-1}^{(\ell)} \tag{3.2}$$

$$p_{S,k|k-1}^{(\ell)} = \sum_{j=1}^{J} \omega_{k|k-1}^{(j)} \mathcal{N}\left(x; \bar{x}^{(j)}, P^{(j)}\right). \tag{3.3}$$

The update equations in Eqs. 2.40-2.42 are similarly updated with the GMM assumption as follows:

$$\eta_Z^{(\theta)}(\ell) = \sum_{j=1}^{J} \omega_{k,Z}^{(j,\theta)}(\ell) \tag{3.4}$$

$$p^{(\theta)}(x, \ell | Z) = \sum_{j=1}^{J} \frac{\omega_{k,Z}^{(j,\theta)}}{\eta_Z^{(\theta)}(\ell)} \mathcal{N}\left(x; x_{k,Z}^{(j,\theta)}(\ell), P_{k,Z}^{(j,\theta)}(\ell)\right) \tag{3.5}$$

where

$$\omega_{k,Z}^{(j,\theta)} = \omega_{k|k-1}^{(j)} \begin{cases} \frac{p_D(\ell) q_j(z_{\theta(\ell)}; \ell)}{\kappa(z_{\theta(\ell)})}, & \text{if } \theta(\ell) > 0 \\ 1 - p_D(\ell), & \text{if } \theta(\ell) = 0 \end{cases} \tag{3.6}$$

$$q_j(z; \ell) = \mathcal{N}\left(y, h\left(x_{k|k-1}^{(j)}(\ell)\right), P_{zz}^{(j)}(\ell)\right) \tag{3.7}$$

$$x_{k,Z}^{(j,\theta)}(\ell) = \begin{cases} x_{k|k}^{(j)}(z_{\theta(\ell)}), & \text{if } \theta(\ell) > 0 \\ x_{k|k-1}^{(j)}(\ell), & \text{if } \theta(\ell) = 0 \end{cases} \tag{3.8}$$

$$P_k^{(j,\theta)}(\ell) = \begin{cases} P_{k|k}^{(j)}(\ell), & \text{if } \theta(\ell) > 0 \\ P_{k|k-1}^{(j)}(\ell), & \text{if } \theta(\ell) = 0 \end{cases} \tag{3.9}$$

where $h(x)$ is some appropriate measurement model, $x_{k|k}^{(j)}$ and $P_{k|k}^{(j)}$ are the updated mean and covariance of each component, respectively, and p_D is the probability of detection. The mean and covariance, $x_{k|k}^{(j)}$ and $P_{k|k}^{(j)}$, are updated

utilizing the single target update equations of the unscented Kalman filter, specifically Eqs. 2.18-2.21.

3.2 Rényi and Cauchy-Schwarz Divergence

The Rényi and Cauchy-Schwarz divergences were chosen to train the reinforcement learning algorithm. The Rényi divergence does not have a closed-form solution for Gaussian mixtures and instead a numerical integration is required. The Cauchy-Schwarz divergence does have a closed form solution, and computes much faster than the Rényi divergence. If all of the targets are not visible in the sensor field of regard at all times, the Cauchy-Schwarz divergence is not a viable metric for the reward function [9].

3.2.1 Rényi Divergence Implementation

The Rényi divergnce has not yet been extended to the LMB filter, however, formulations exist for the Probability Hypothesis Density (PHD) filter, which is another class of RFS filters [51]. The conversion from the LMB filter state to the PHD filter state is trivial. The LMB state is parameterized by $\{r^{(\ell)}, p^{(\ell)}(x)\}_{\ell \in \mathbb{L}}$ and can be transformed into a PHD state by

$$v(x) = \sum_{\ell \in \mathbb{L}} r^{(\ell)} p^{(\ell)}(x) \tag{3.10}$$

where $v(x)$ is the expected number of targets [42]. The PHD is formulated to have cardinality distributions represented as Poisson distributions. The PHD

state then becomes

$$p(x) = e^{-\lambda_0}\lambda_0 \prod_{x \in X} s_0(x) \tag{3.11}$$

$$q(x) = e^{-\lambda_1}\lambda_1 \prod_{x \in X} s_1(x) \tag{3.12}$$

where $p(x)$ and $q(x)$ are the probability distributions found in Eq. 2.48, λ is the mean of the Poisson distribution, and $s(\cdot)$ is some representation of the spatial pdf. Then substituting Eqs. 3.11 and 3.12 into Eq. 2.48 and invoking the identity $e^x = \sum_{n=0}^{\infty} x^n/n!$ which transforms the Rényi divergence to

$$D_\alpha = \lambda_0 + \frac{\alpha}{1-\alpha}\lambda_1 + \frac{\lambda_1^\alpha \lambda_0^{1-\alpha}}{\alpha - 1} \int s_1(x)^\alpha s_0^{1-\alpha} dx. \tag{3.13}$$

Since the spatial pdf, $s(\cdot)$, is represented as the sum of Gaussian mixtures, Eq. 3.13 is then approximated by replacing $s(\cdot)$ with the Gaussian mixtures and replacing λ with the sum of the weights of the Gaussian mixtures. As dicussed in Chapter 2, the largest discrimination between two pdfs is found by setting α to 0.5, and with these assumptions, Eq. 3.13 becomes

$$D_\alpha \approx \sum_i w_i^0 + \sum_i w_i^1 - 2 \int \left(\sum_i \sum_j \mathcal{N}(x; m_i^0, P_i^0)\mathcal{N}(x; m_i^1, P_j^1) \right)^{0.5} dx \tag{3.14}$$

where w is the weight of the corresponding Gaussian distribution, m is the mean of the same distribution, and P is the covariance of the same distribution. Unfortunately, the integral in Eq. 3.14 does not have a closed-form solution and importance sampling must be employed to produce a solution. To facilitate the use of importance sampling, the two sums of Gaussian distributions found in

Eq. 3.14 dependent on x is manipulated to only hold a single sum of Gaussian distributions dependent on x. The result of this manipulation is

$$D_\alpha \approx \sum_i w_i^0 + \sum_i w_i^1 - 2\int \left(\sum_i \sum_j \mathcal{N}(m_i^0; m_j^1, P_i^0 + P_j^1)\mathcal{N}(x; m_{i,j}, P_{i,j}) \right)^{0.5} dx \qquad (3.15)$$

where

$$m_{i,j} = P_{i,j}\left((P_i^0)^{-1}m_i + (P_j^1)^{-1}m_j\right)$$
$$P_{i,j} = \left((P_i^0)^{-1} + (P_j^1)^{-1}\right)^{-1}.$$

With the form presented in Eq. 3.15, importance sampling is utilized with samples generated from the second distribution and evaluated against the first.

3.2.2 Cauchy-Schwarz Divergence Implementation

The Rényi divergence computation proved to be a barrier to quickly find the necessary tuning parameters required for training. Since all of the targets were in the field of regard of the sensor, the Cauchy-Schwarz divergence was chosen as an alternative.

The Cauchy-Schwarz divergence is not dependent on the representation of the pdf, but holds a closed-form solution when the pdf is represented as a sum of Gaussian mixtures. Using the same Poisson distributions in Eqs 3.11 and 3.12 to represent the cardinality distribution and then again the sum of

Gaussin mixtures represent $s(\cdot)$, the following closed-form solution is found

$$\begin{aligned} D_{CS} =& \frac{1}{2}\sum_i \sum_j w_i^0 w_j^0 \mathcal{N}(m_i^0; m_j^0, P_i^0 + P_j^0) \\ &+ \frac{1}{2}\sum_i \sum_j w_i^1 w_j^1 \mathcal{N}(m_i^1; m_j^1, P_i^1 + P_j^1) \\ &- \sum_i \sum_j w_i^0 w_j^1 \mathcal{N}(m_i^0; m_j^1, P_i^0 + P_j^1) \end{aligned} \tag{3.16}$$

where w, m, and P, correspond to the appropriate Gaussian distribution in the sum of the Gaussian mixtures [16].

3.3 Reinforcement Learning Implementation

The algorithms implemented in this work were the PGQL algorithm and the Q-learning algorithm prescribed in [38]. The PGQL algorithm was chosen as the primary on-policy method and Q-learning was chosen as the primary off-policy method. While the PGQL algorithm is "slightly off-policy" [45], it is still required to be on-policy due to the actor-critic algorithm embedded in it. The PGQL algorithm was chosen due its higher performance in most Atari 2600 games when compared with the A3C and DQN algorithms and DQN was chosen due to its higher rate of use in multi-agent systems. Additionally, Linares, et. al. [33, 32], has explored earlier on-policy methods.

3.3.1 Policy Gradient Q-Learning Implementation

The PGQL algorithm utilizes an actor-critic algorithm to quickly converge on a feasible solution and Q-learning to explore the solution space to pre-

vent a local maxima from being found. The PGQL algorithm finds a connection between the actor-critic methods utilizing a soft-max output with entropy regularization and Q-learning. The soft-max function is discussed in Sec. 3.4.1 and entropy regularization is used to induce exploration during training by adding an entropy penalty to the cost function. O'Donoghue, et. al. [45], shows that it is possible to combine the actor and critic networks to recover the Q-function, which is possible due to equivalency of actor-critic methods and action-value methods. The Q-function, as a function of the actor network and critic network is then

$$Q(s,a) = \lambda \left(\log \pi_\theta(s,a) - \sum_a \pi_\theta(s,a) \log \pi_\theta(s,a) \right) + V_\phi(s) \tag{3.17}$$

where λ is the entropy regularization factor, π_θ is the actor network parameterized by θ, V_ϕ is the critic network parameterized by ϕ, and s and a refer to the state and action, respectively. If the true value, or critic, function was known then the update equations found in [45] could be utilized. However, for complex problems, the true critic is not known prior to training and instead a more practical implementation is utilized for on-line learning.

Instead, the agent, or in this case, the sensor, interacts with the environment and takes actions and recieves rewards using an actor-critic paradigm. The cost function to minimize for the each the actor (Eq. 3.18) and critic (Eq. 3.19) are

$$\begin{aligned} J(\theta) = -\mathbb{E}\left[\log \pi_\theta(r(s,a) + \gamma V_{\phi^-}(s') - V_\phi(s))\right] - \\ \lambda \sum_a \pi_\theta(s,a) \log \pi_\theta(s,a) \end{aligned} \tag{3.18}$$

$$J(\phi) = \mathbb{E}\left[r(s,a) + \gamma V_{\phi^-}(s') - V_\phi(s)\right]^2 \tag{3.19}$$

where γ is the discount factor for infinite horizon MDP, s' is the next state, and ϕ^- refers to the network at some previous instance. At each instance an action is taken, the tuple $\{s, a, r, s'\}$ is stored for use in Q-learning.

The Q-learning aspect of the PGQL algorithm is similar to the DQN method proposed in [38], but instead of operating on a Q-network the actor and policy networks are updated by taking the gradient of the loss function in Eq. 3.20 with respect to each of the function parameterizations, θ and ϕ. The loss function for the Q-learning aspect is

$$J_Q(\theta, \phi) = \mathbb{E}\left[Q(s,a) - \left(r(s,a) + \gamma \max_{a'} Q^-(s', a)\right)\right]^2 \tag{3.20}$$

where Q^- is some previous target Q-network. Then the gradients are

$$\nabla_\theta J = \mathbb{E}\left[Q(s,a) - \left(r(s,a) + \gamma \max_{a'} Q^-(s', a)\right)\right] \nabla_\theta \log \pi(s,a) \tag{3.21}$$

$$\nabla_\phi J = \mathbb{E}\left[Q(s,a) - \left(r(s,a) + \gamma \max_{a'} Q^-(s', a)\right)\right] \nabla_\phi V(s) \tag{3.22}$$

which are used to update the actor and critic networks.

During testing, it was found that a single pass through was not enough interactions to train via the algorithm, and so instead each policy is tested over 20 iterations of varying, but similar, data. The data is varied by perturbing the initial states and also by generating new noisy measurements each iteration. The increase in the batch size proved to be key in stabilizing the algorithm. Additionally, since both the Rényi and Cauchy-Schwarz divergences range from 0 to ∞, the rewards were normalized with mean zero and unit variance for the

actor-critic batch for the 20 iterations. Additionally, the random sampling in the replay buffer was normalized after the batch was chosen.

3.3.2 Q-Learning Implementation

The Q-learning algorithm does not have an actor network to generate an action and so is chosen instead by choosing the action that corresponds to the largest state value. This greedy approach does not lend itself to exploration and instead of entropy regularization, a random action is chosen with with some probability ϵ. Different versions of varying ϵ exist, but a linear schedule that decreases with number of samples is used. Much of the implementation for the Q-learning implementation follows directly from the PGQL implementation.

3.4 Neural Network Design

The PGQL algorithm and the Q-learning algorithm require different numbers of neural networks and different output sizes. In this context, both neural networks utilized hyperbolic tangents for activation functions due to the speed of convergence in the PGQL algorithm with the ReLU activation functions. Neural network training performs best when the inputs are whitened, i.e., have a mean of zero and a standard deviation of one [27]. The inputs are whitened for both the PGQL and the Q-learning algorithms. The neural networks and reinforcement learning algorithms were developed using Google's TensorFlow library [1].

3.4.1 PGQL Neural Network Design

The PGQL neural networks are a set of four neural networks, but only two are trained and utilized to generate the policy. The networks are constructed in a similar fashion to the ones found in Linares, et. al. [33, 32]. Specifically, the actor network, which represents the policy, was constructed with three hidden layers with 200 neurons, 100 neurons, and 50 neurons, respectively. The input neurons have dimensions $6 \cdot n$, where n is the number of targets and output neurons are the potential actions and are of size n as well. The hidden layers activation function are hyperbolic tangents, and the output layer do not have activation functions. The actor network's outputs are later passed into the soft-max function. The soft-max function is defined as

$$P(x_j) = \frac{e^{x_j}}{\sum_{i=1}^{n} e^{x_i}} \quad \forall j = 1, ..., n \tag{3.23}$$

and provides a probability that can be thought of as the action with the highest reward. The critic network is constructed identically except for the output neurons which is of size one.

The PGQL algorithm requires target networks so that the actor and critic networks are not bootstrapped (causing instability). The two additional networks are identical in dimension and size as the actor and critic networks described previously. As discussed in Section 3.3, the target networks are periodically updated by copying the weights and biases of the actively trained actor and critic networks.

3.4.2 Q-Learning Network Design

The Q-learning algorithm only requires two networks, instead of the four for the PGQL algorithm. The Q-learning neural network is identical to the PGQL actor network in that it has $6 \cdot n$ input neurons, and then 3 hidden layers with 200 neurons, 100 neurons, and 50 neurons, respectively, and finally n output neurons. However, the soft-max function is not utilized here, and instead the raw values are used as the Q-values. Again, the second network is utilized as the target network and is updated infrequently by copying the weights and biases of the trained network.

3.5 Simulation

There are two different scenarios to test the reinforcement learning algorithms. The first is a simple 2 dimensional problem to see if the PGQL algorithm is able to train in a multi-sensor domain and the second is a test of the interactions of the reinforcement learning algorithms alongside the LMB filter. Each require specific tuning parameters, which for ease, are tabulated in Tables 3.1 and 3.2.

Table 3.1: Reinforcement learning and neural network tuning parameters for the space situational awareness case.

Parameter	PGQL	Q-learning
Discount Factor, γ	0.99	0.99
Exploration Factor, λ	100 or 0.1	N/A
Initial ϵ	N/A	0.9
Final ϵ	N/A	0.1
Actor Learning Rate	$1e-3$	N/A
Critic Learning Rate	$1e-3$	N/A
Actor Q-Learning Rate	$1e-3$	N/A
Critic Q-Learning Rate	$1e-3$	N/A
Q-Learning Rate	N/A	$1e-3$
Q-Learning Minibatch Size	50,000	50.000
Min. Steps for Q-Learning	50,000	50,000
Max. Buffer Size	1e6	1e6
Target Update Freq.	100	100

Table 3.2: Reinforcement learning and neural network tuning parameters for the space situational awareness case.

Parameter	PGQL	Q-learning
Discount Factor, γ	0.99	0.99
Exploration Factor, λ	1000	N/A
Initial ϵ	N/A	0.9
Final ϵ	N/A	0.1
Actor Learning Rate	$1e-3$	N/A
Critic Learning Rate	$1e-3$	N/A
Actor Q-Learning Rate	$1e-3$	N/A
Critic Q-Learning Rate	$1e-3$	N/A
Q-Learning Rate	N/A	$1e-3$
Q-Learning Minibatch Size	100	100
Min. Steps for Q-Learning	100	100
Max. Buffer Size	500	500
Target Update Freq.	200	200

3.5.1 Linear 2-Dimensional Problem

The problem utilizes a linear, Gaussian Kalman filter with three targets. The target dynamics for each target are

$$\dot{\boldsymbol{x}} = \underbrace{\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\boldsymbol{A}} \boldsymbol{x} \tag{3.24}$$

where the state, $\boldsymbol{x}$, is a 4x1 vector representing position and velocity. Eq. 3.24 gives us the state transition matrix, $\boldsymbol{\Phi}$, as

$$\boldsymbol{\Phi}(t) = \begin{bmatrix} 1 & 0 & t & 0 \\ 0 & 1 & 0 & t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.25}$$

where t is time. Additionally, the observations the sensor can make are limited to either the x-component of the state or the y-component depending on where the sensors are positioned. The linear observation model for each sensor type is

$$H_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}^T \qquad H_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}^T \tag{3.26}$$

Sensor positions and initial starting points of the three targets are seen in Fig. 3.3. Sensor noise is different for each to induce heterogeneity and for sensor 1 is 0.01 meters^2 and for sensor 2 is 0.04 meters^2. The true initial states and covariances can be found in Table 3.3.

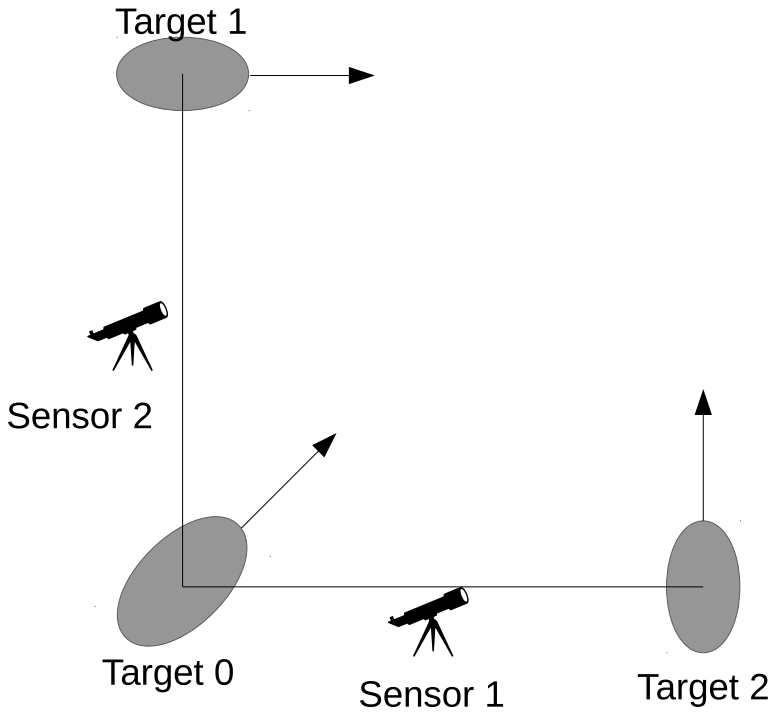

Figure 3.3: Initial starting conditions and sensor setup for the linear 2-D problem.

Table 3.3: Initial conditions for the 2-D problem.

Target #	State	Covariance
0	$\begin{bmatrix} 0 \text{ m} \\ 0 \text{ m} \\ 10 \text{ m/s} \\ 10 \text{ m/s} \end{bmatrix}$	diag $\begin{bmatrix} 1\text{e}-2 \text{ m}^2 \\ 1\text{e}-2 \text{ m}^2 \\ 2\text{e}-4 \text{ (m/s)}^2 \\ 2\text{e}-4 \text{ (m/s)}^2 \end{bmatrix}$
1	$\begin{bmatrix} 100 \text{ m} \\ 0 \text{ m} \\ 0 \text{ m/s} \\ 10 \text{ m/s} \end{bmatrix}$	diag $\begin{bmatrix} 1\text{e}-2 \text{ m}^2 \\ 1\text{e}-2 \text{ m}^2 \\ 1\text{e}-4 \text{ (m/s)}^2 \\ 1\text{e}-4 \text{ (m/s)}^2 \end{bmatrix}$
2	$\begin{bmatrix} 0 \text{ m} \\ 100 \text{ m} \\ 10 \text{ m/s} \\ 0 \text{ m/s} \end{bmatrix}$	diag $\begin{bmatrix} 1\text{e}-2 \text{ m}^2 \\ 1\text{e}-2 \text{ m}^2 \\ 2\text{e}-4 \text{ (m/s)}^2 \\ 2\text{e}-4 \text{ (m/s)}^2 \end{bmatrix}$

3.5.2 Ten Target Geosynchronous Problem

The ten target case requires more advanced dynamics and the use of the LMB filter along with the reinforcement learning. The toy problem's use case was to test the feasibility of an on-policy policy gradient algorithm in a multi-agent scenario. The ten target case's goal is to check viability of a multi-target filter working in conjunction with reinforcement learning.

The scenario uses two-body dynamics defined in Eq. 3.27 to propagate the states [60].

$$\ddot{\boldsymbol{r}} = -\frac{\mu}{||\boldsymbol{r}||_2^2}\frac{\boldsymbol{r}}{||\boldsymbol{r}||} \tag{3.27}$$

where μ is the gravitational constant of the Earth, and $\boldsymbol{r}$ is a vector of the Cartesian position.

The sensor for this problem is placed at 20.71° Latitutde, −156.6° Longitude, 3058.6 m above the nominal Earth's radius and is modeled as an electro-optical sensor. An electro-optical sensor typically gives right ascension and declination observations of satellites from Cartesian state inputs which are given as

$$RA = \arctan \frac{y - y_s}{x - x_s} \tag{3.28}$$

$$Dec = \arcsin \frac{z - z_s}{||\boldsymbol{r} - \boldsymbol{r}_s||} \tag{3.29}$$

where x, y, z are the corresponding components of the Cartesian state $\boldsymbol{r}$, x_s, and y_s, z_s are the corresponding components of the sensor state $\boldsymbol{r_s}$. The sensor is assumed to have a field of view in the shape of a square of size 0.5° in both right ascension and declination. The sensor is also assumed to have a sensor noise of 1 arcsecond in both right ascension and declination.

The initial state pdf of the targets can be found in [17]. As a summary, they are based on the EchoStar1 spacecraft with orbital uncertainties such that the variances the sum of the standard deviations are approximately 250 m in position and 0.05 m/s velocity [17]. The exact iniitial states can be found in Table A.1. While, no target is assumed to disappear, a probability of survival and existence less than one is required to prevent singularities. As such, the probability of survival is set to 0.999, the probability of existence is set to 0.9999, and the probability of detection for the sensor is set 1.

Chapter 4

Results

The focus of this chapter is to use the ideas and implementations from Chs. 2 and 3 and show the performance of the algorithms in the sensor tasking domain. The goal of each simulation was to show a different potential capability of the reinforcement learning algorithms by introducing them to an RFS-based filter with the LMB filter and their robustness in multi-sensor cases. This chapter is organized by first discussing the results of a single sensor agent interacting with space objects in geosynchronous orbits and then followed by a brief look into the multi-sensor cases.

4.1 Geosynchronous Orbit Tasking Results

Many different scenarios were tested to show both the capabilities of the PGQL and Q-learning algorithms. First, a comparison of the Rényi divergence and Cauchy-Schwarz divergence due to the computational effort required in the former. The exploration factor, α, in the entropy regularization was found to play a large role in the overall performance of the networks and so α was varied as well. Additionally, a comparison of Q-learning, an off-policy learning technique, and PGQL is shown for the single agent case.

4.1.1 Comparison of Reward Function

The numerical integration of the Rényi divergence caused the neural network training to take, on average, around 40 hours. Instead, the Cauchy-Schwarz divergence, for the same training scenario, would complete, on average, around 7 hours. If all of the targets are in the field of regard of the sensors, then both reward functions should give similar results and thus the Cauchy-Schwarz divergence can be used instead of the Rényi divergence. The ten geosynchronous space objects were intialized such that during the course of the night, a sensor positioned at Maui, HI is able to capture all of the targets, when the sensor has a square field of view of 2° in right ascension and declination [17].

4.1.1.1 Rényi Divergence Results

The Rényi divergence utilizes the same neural network defined in Sec. 3.4.1 with λ, the exploration factor, set to 100. First, to confirm that the algorithm was successful, the Mahalanobis distance is utilized to confirm that the filter is consistent and is able to stay within the 3σ bounds in terms of uncertainty. The Mahalanobis distance is defined as

$$D_M = \sqrt{(\boldsymbol{x} - \bar{\boldsymbol{x}})^T \boldsymbol{P}^{-1} (\boldsymbol{x} - \bar{\boldsymbol{x}})} \tag{4.1}$$

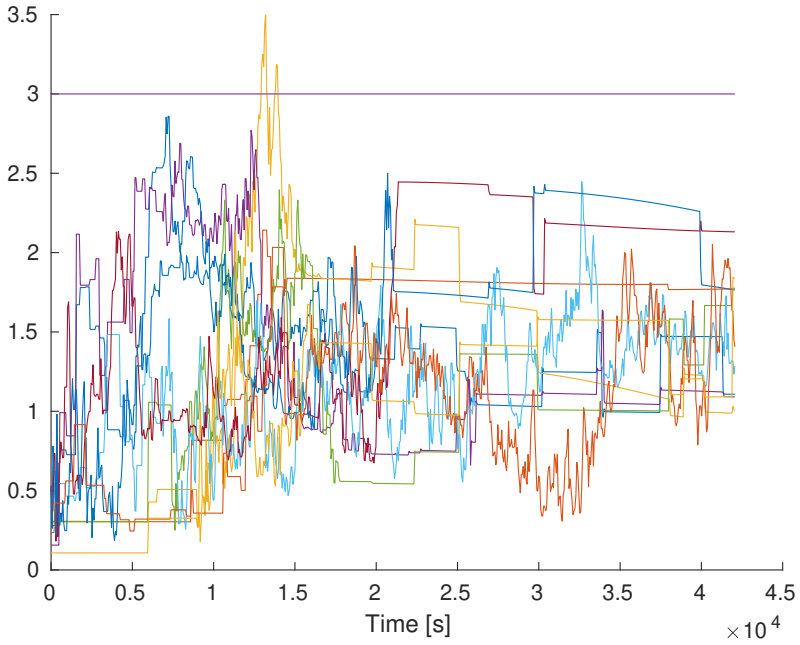

Figure 4.1: The Mahalanobis distance for the Rényi divergence for the full state over the entire night for all 10 targets.

A Mahalanobis distance under three is preferred, but slight peaks above three is valid for filter consistency. The filter is still attempting to converge to a solution early on and so peaks above three can be seen then. The Mahalanobis distance is unitless, however, due to the different units in the state and covariance, the full state Mahalanobis distance is not unitless. Due to this, the position and velocity Mahalanobis distances were calculated to further validate the performance of the filter.

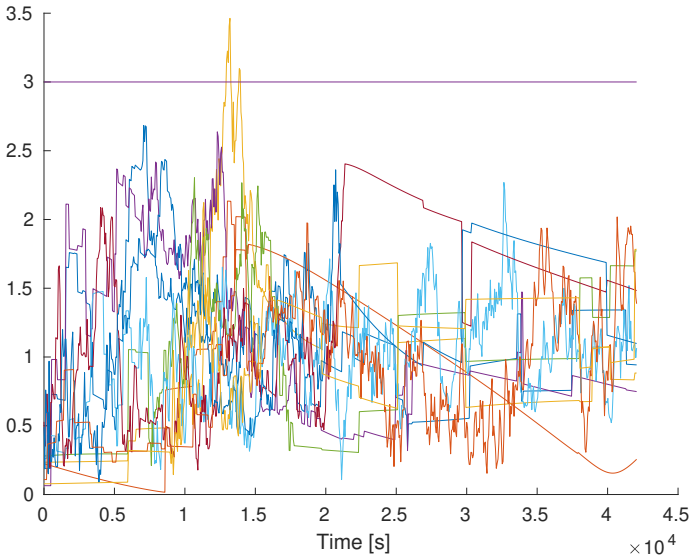

Figure 4.2: The Mahalanobis distance for the Rényi divergence for just the position states over the entire night for all 10 targets.

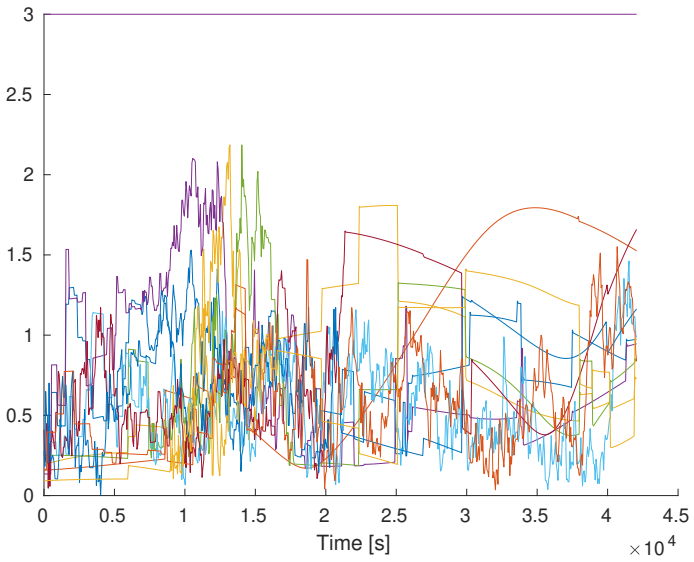

Figure 4.3: The Mahalanobis distance for the Rényi divergence for just the velocity states over the entire night for all 10 targets.

The peak over three in Fig. 4.1 is caused by the position error and uncertainty. Additionally, from the continuity and discontinuity of the lines, the targets are observed and not observed, respectively. In terms of individual targets, the largest uncertainty throughout the night is seen in Target 2 which increases to a maximum of about 5 km 3σ in the x-position component, but quickly collapses after a single measurement as seen in Fig 4.4. The other targets do not see such a large uncertainty in any component.

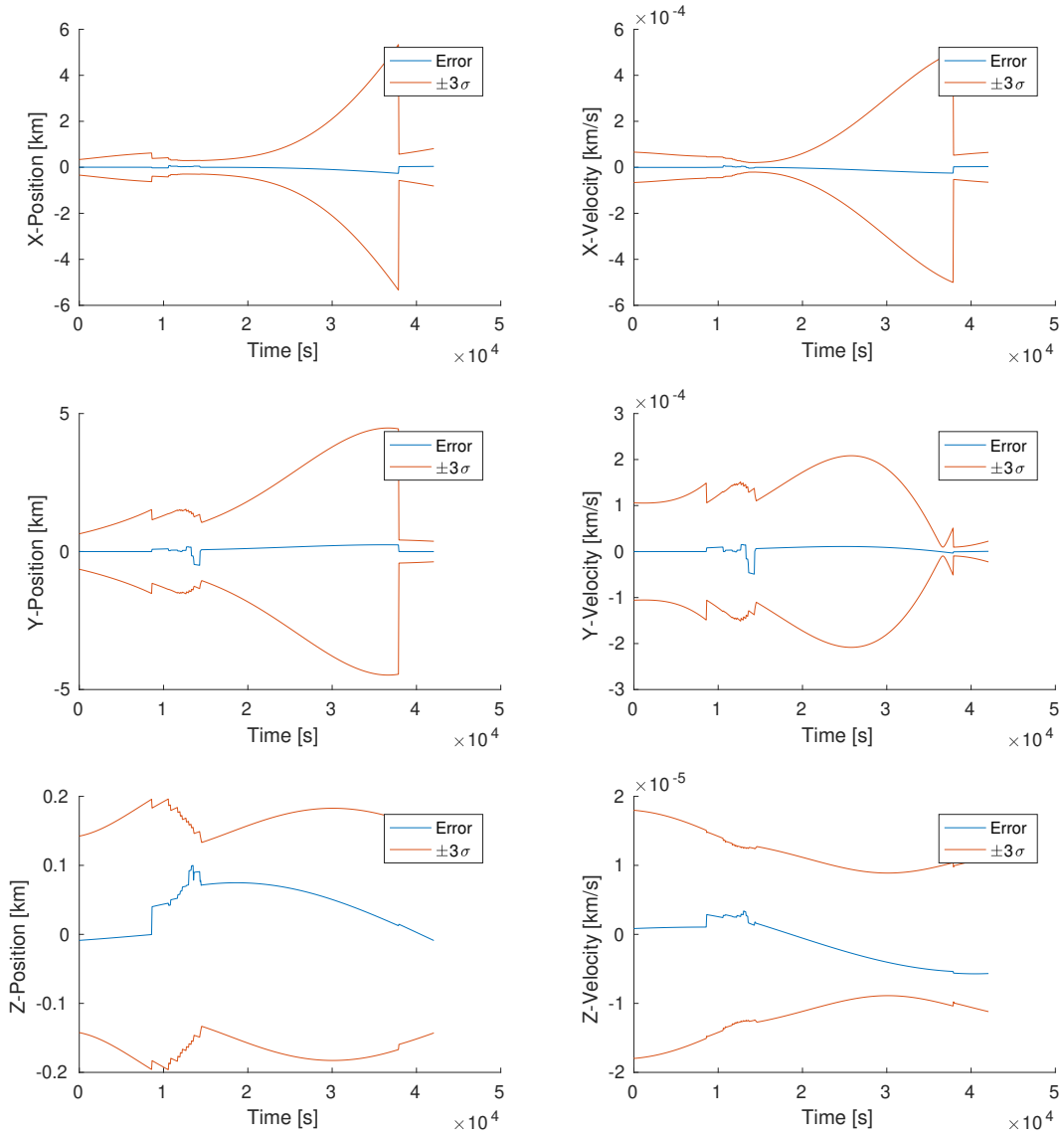

Figure 4.4: Target 2 Errors with $\pm 3\sigma$ envelopes for the Rényi divergence.

The final measure of performance is the optimal subpattern assignment (OSPA) error. The OSPA error was first developed by Schuhmacher, et. al. [54], as an error bound that was consistent for multi-target filtering. The

OSPA error is calculated seperately for both the position and the velocity due to units mismatching. The OSPA error is defined as

$$d_{OSPA}^{(c)}(\boldsymbol{X}, \boldsymbol{Y}) = \left(\frac{1}{n} \left(\min_{\pi \in \Pi_n} \sum_{i=1}^{m} d^{(c)}(\boldsymbol{x}_i, \boldsymbol{y}_{\pi(i)})^p + c^p(n-m) \right) \right)^{1/p} \tag{4.2}$$

where X and Y are the finite subsets that correspond to the individual realizations at some time t, x and y are those realizations that belong to sets X and Y, π is some permutation belong to set Π_n which contains a set of permutations of size n, n and m refer to the sizes of the finite subset X and Y, respectively, c is the cutoff distance, and p is the type of p-norm. The distance $d^{(c)}(\boldsymbol{x}_i, \boldsymbol{y}_{\pi(i)})$ is defined as

$$d^{(c)}(\boldsymbol{x}_i, \boldsymbol{y}_{\pi(i)}) = \min\left(c, d(\boldsymbol{x}, \boldsymbol{y})\right) \tag{4.3}$$

where $d(\cdot, \cdot)$ is some distance metric.

For the Rényi divergence case, the OSPA error plots over time can be found in Figs. 4.5 and 4.6. The OSPA error c and p values are set to 100 km and 2, respectively.

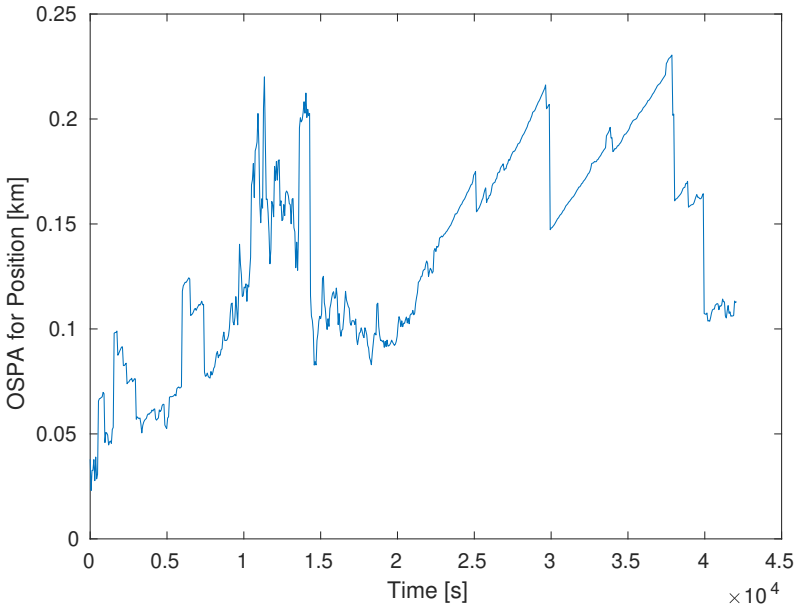

Figure 4.5: Position OSPA error for the Rényi divergence.

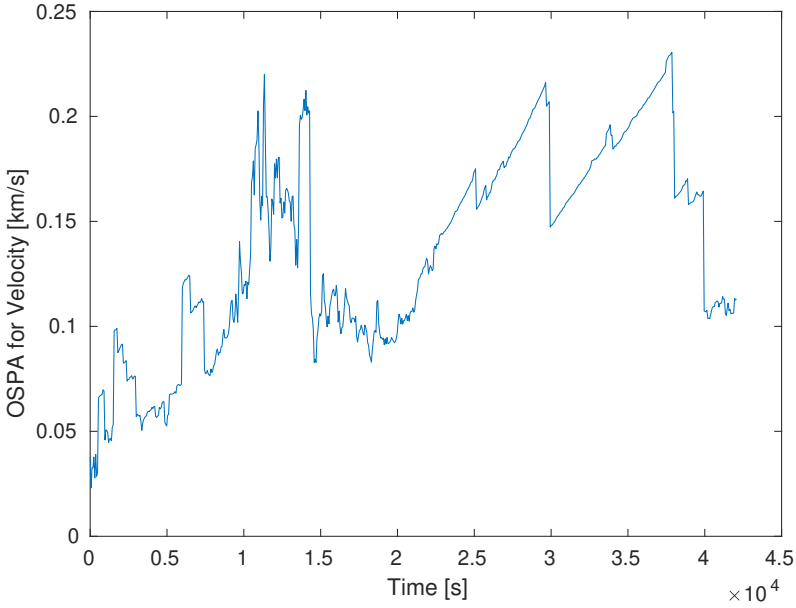

Figure 4.6: Velocity OSPA error for the Rényi divergence.

The OSPA errors are fairly consistent throughout time and with observations to targets with long delays of another observation, the OSPA error quickly drops. With this scenario, a baseline for subsequent results has been established.

4.1.1.2 Cauchy-Schwarz Divergence Results

The Cauchy-Schwarz divergence case is identical to the Rényi divergence case, except the reward function. Again, starting with the Mahalanobis distance:

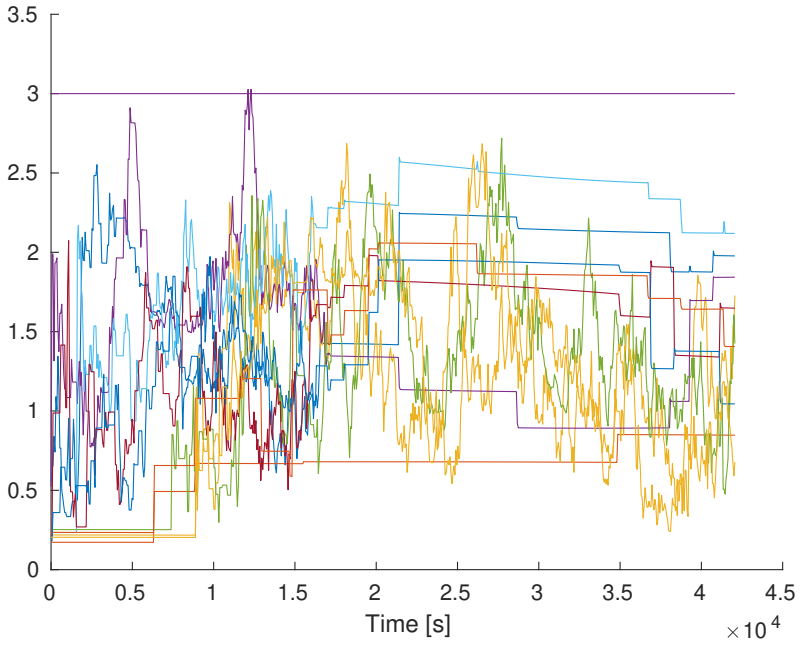

Figure 4.7: The Mahalanobis distance for the Cauchy-Schwarz divergence for the full state over the entire night for all 10 targets.

With only a slight increase above three for a single target, the filter stays

consistent over the course of the night in Fig. 4.7. Additionally, the position and velocity Mahalanobis distances are also shown in Figs. 4.8 and 4.9.

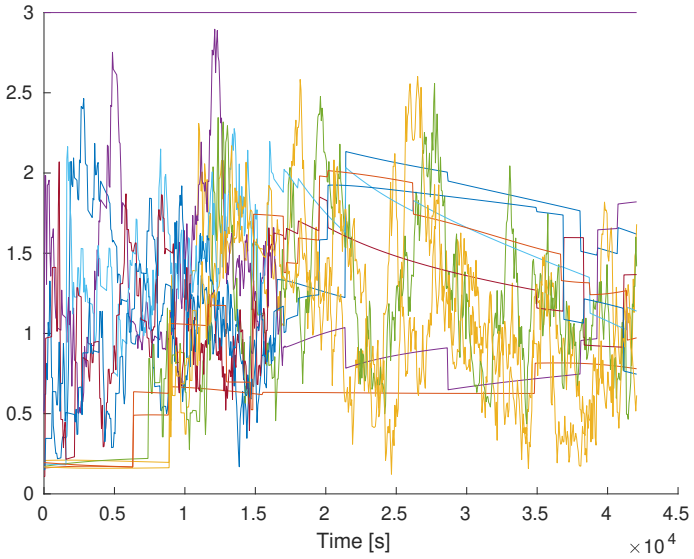

Figure 4.8: The Mahalanobis distance for the Cauchy-Schwarz divergence for just the position states over the entire night for all 10 targets.

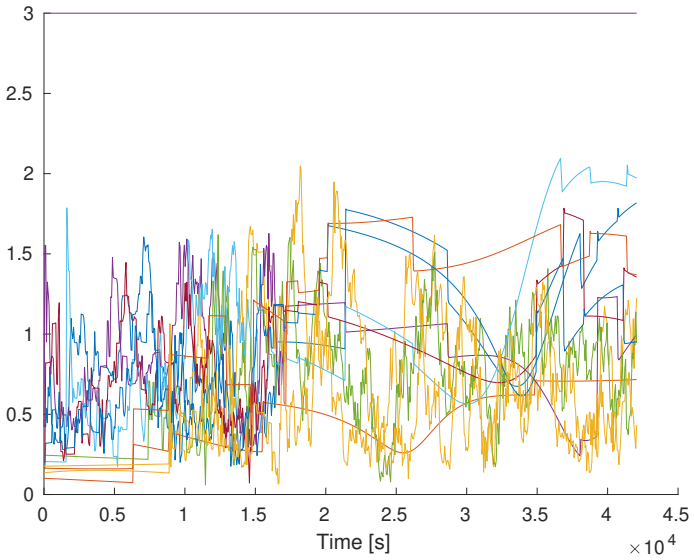

Figure 4.9: The Mahalanobis distance for the Cauchy-Schwarz divergence for just the velocity states over the entire night for all 10 targets.

The peak seen in Fig. 4.7 is caused by the error and uncertainty in the position state as seen in Fig. 4.8. Note that the Mahalanobis distances in Figs. 4.7-4.9 are similar to the Rényi divergence Mahalanobis distances in Figs. 4.1-4.3.

However, the Rényi divergence outperforms the Cauchy-Schwarz divergence in terms of error bounds. Targets 1, 2, 4, and 6-9 all reach an uncertainty in the x-position of about 1.5 km in 3σ. For consistency, the 3σ plots are shown in Fig. 4.10, but the other targets follow a similar structure where measurements are taken near the end of the night.

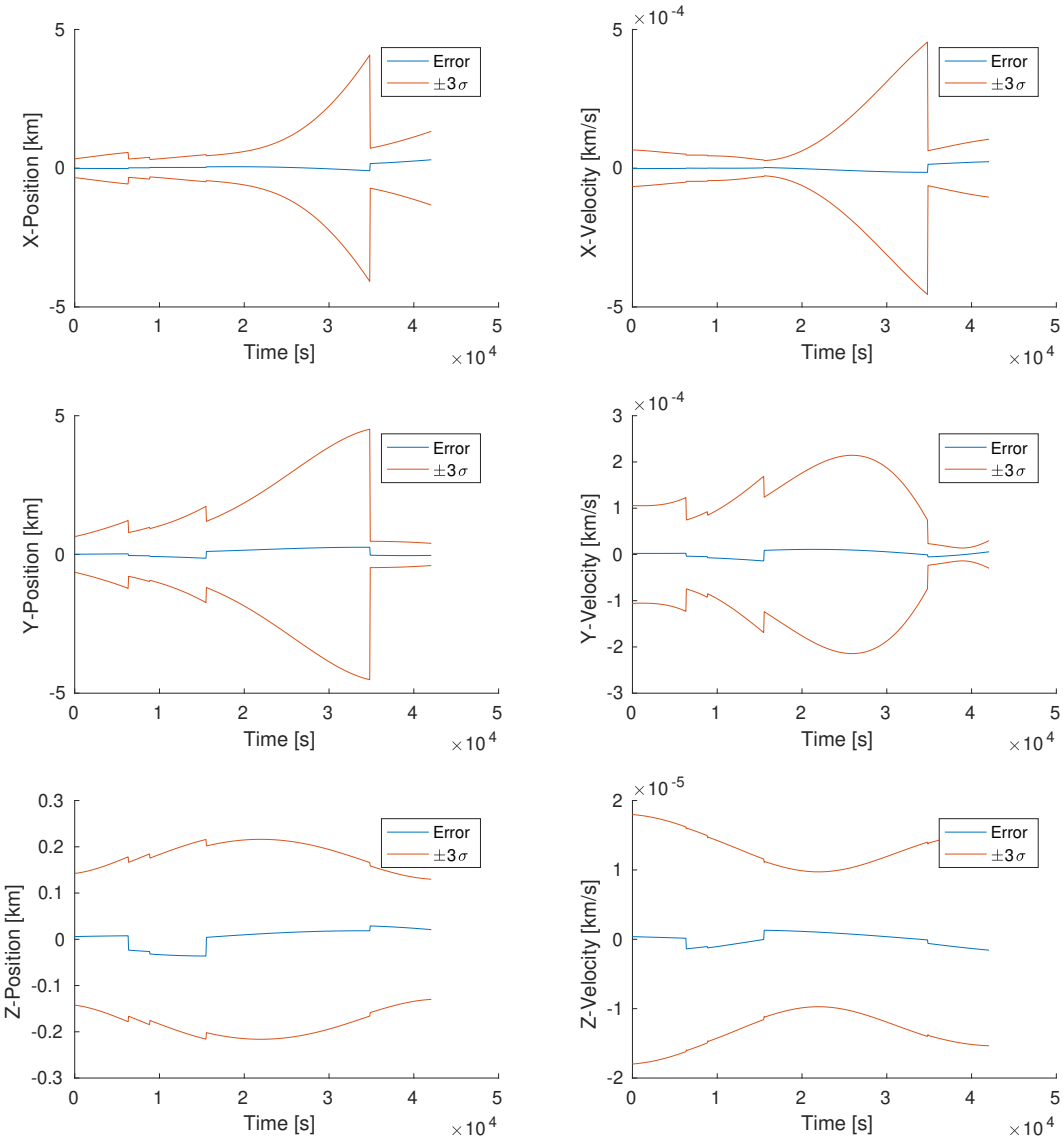

Figure 4.10: Target 2 Errors with $\pm 3\sigma$ envelopes for the Cauchy-Schwarz divergence.

Due to the larger errors and uncertainty in targets 1, 2, 4, 6, 7, 8, and 9, there is also a trend in the OSPA error until the measurements for those

targets later in the night are taken. However, the OSPA error is much higher then the Rényi divergence case as seen in Figs. 4.11 and 4.12.

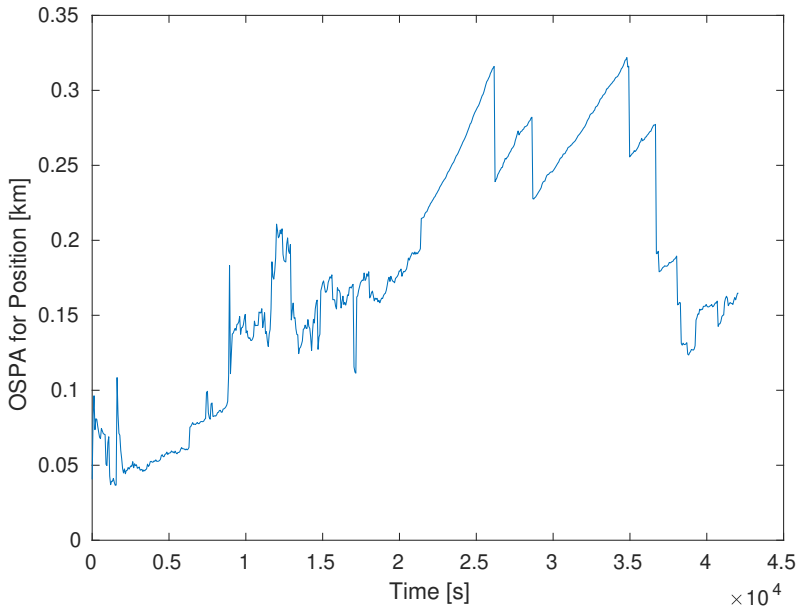

Figure 4.11: Position OSPA error for the Cauchy-Schwarz divergence.

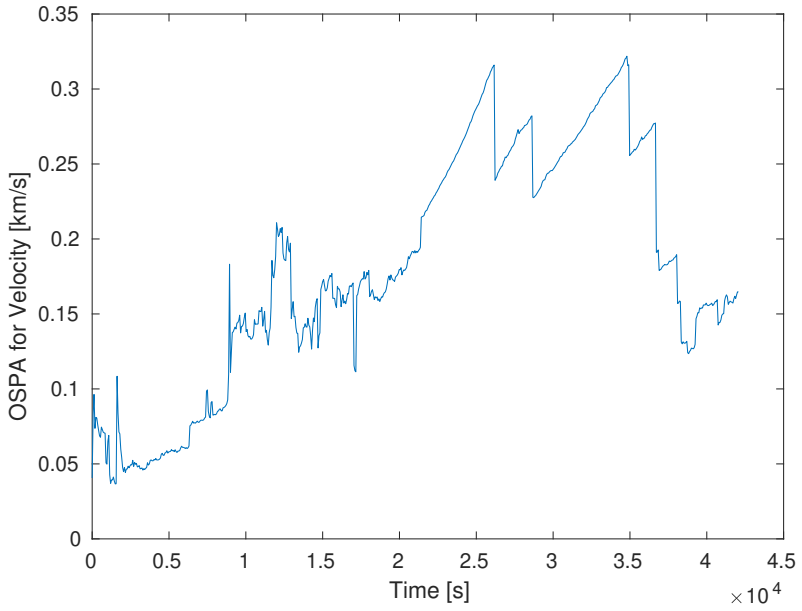

Figure 4.12: Velocity OSPA error for the Cauchy-Schwarz divergence.

While, the OSPA error is higher for a significant portion of the night, it is still able to converge to levels similar the Rényi divergence case.

With this comparison, it seems that the Rényi divergence outperforms the Cauchy-Schwarz divergence, however, it does not seem unreasonable to use the Cauchy-Schwarz divergence in other scenarios.

4.1.2 Exploration Factor Effects on Training

Entropy regularization is an integral part of any policy gradient training paradigm and as such is true for the PGQL algorithm. O'Donoghue, et. al. [45], suggests that the equivalency between actor-critic and action-value training is valid when λ is "relatively small". In Sec. 4.1.1, λ was set to 100,

and so to set λ such that it is "relatively small", in this scenario λ is set to 0.1.

Again, the Mahalanobis distance is utilized to check for filter consistency.

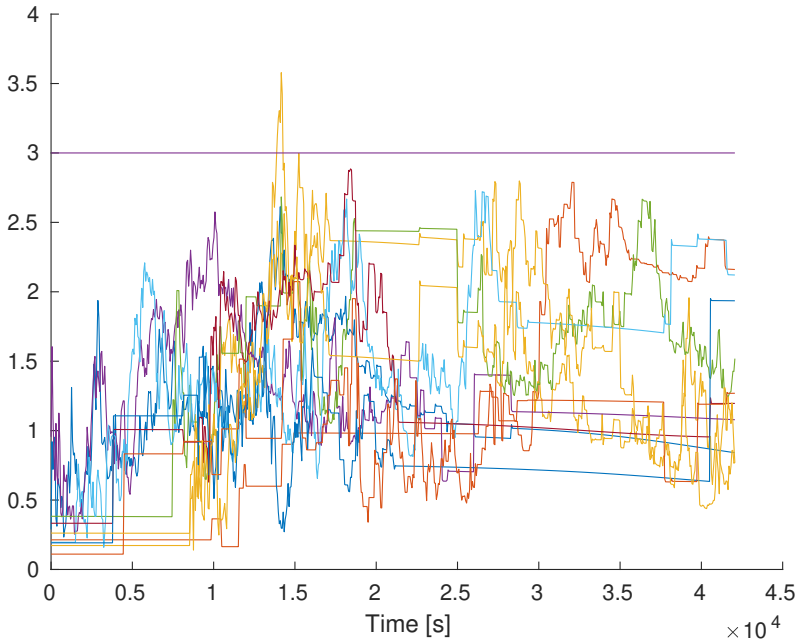

Figure 4.13: The Mahalanobis distance for the Cauchy-Schwarz divergence for the full state over the entire night for all 10 targets with $\lambda = 0.1$.

Fig. 4.13 shows that the filter is consistent. The interesting factor in this test case is that while none of the 3σ bounds increase to the level of target 2 in the previous test case, some targets are observed early in the night and then abandoned as the neural network attempts to exploit the information and rewards it already knows.

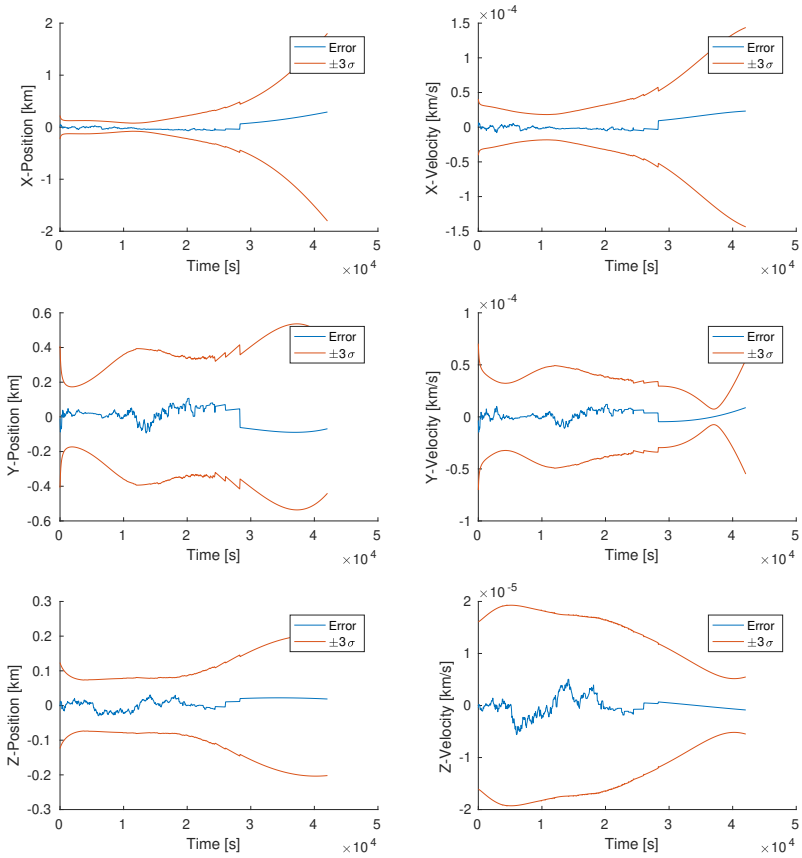

Figure 4.14: Target 1 Errors with $\pm 3\sigma$ envelopes for the Cauchy-Schwarz divergnce with $\lambda = 0.1$.

The uncertainty begins to grow without any measurements in Fig. 4.14 by the end of the night to collapse the uncertainty envelope for the target. By the start of the following night, the uncertainty may have grown so much that the target may be declared lost.

The OSPA also stays consistent with the other scenarios, however, by the end of the night due to the unobserved targets, the OSPA begins to grow.

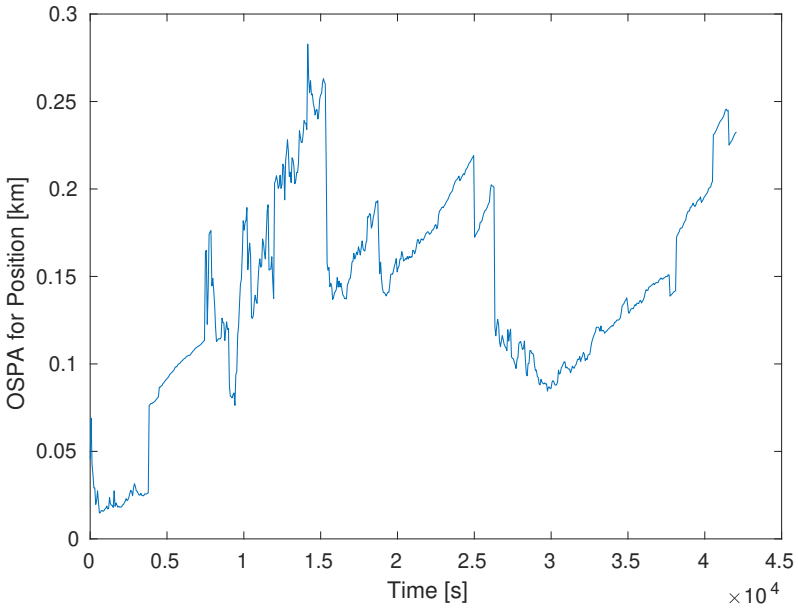

Figure 4.15: Position OSPA error for the Cauchy-Schwarz divergence with $\lambda = 0.1$.

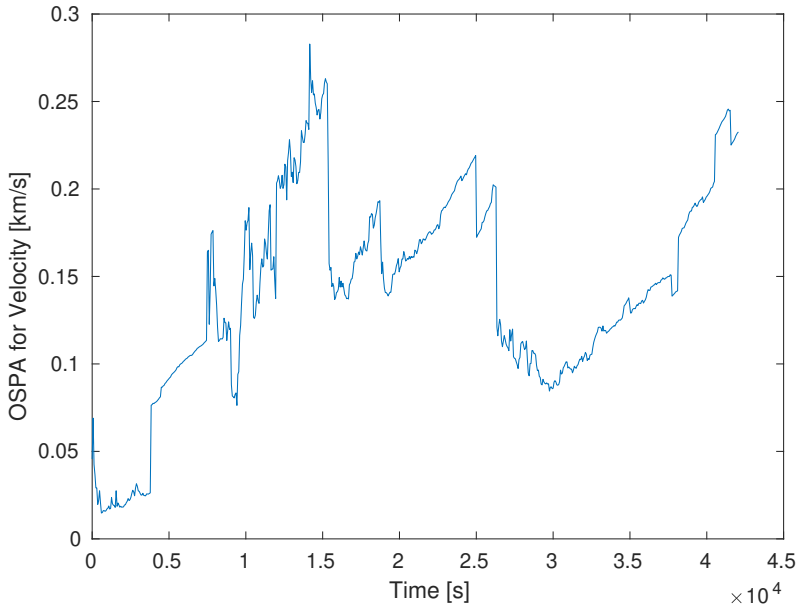

Figure 4.16: Velocity OSPA error for the Cauchy-Schwarz divergence with $\lambda = 0.1$.

4.1.3 Comparison with Off-Policy Methods

Off-policy methods are typically more robust than on-policy methods and so a comparison of the two is performed utilizing the same number of interactions with the environment. The off-policy method used the Q-learning described in Sec. 3.3.2 and can be compared with the Cauchy-Schwarz PGQL case in Sec. 4.1.1.2.

As with the previous scenarios, the Mahalanobis distance is checked to confirm filter consistency.

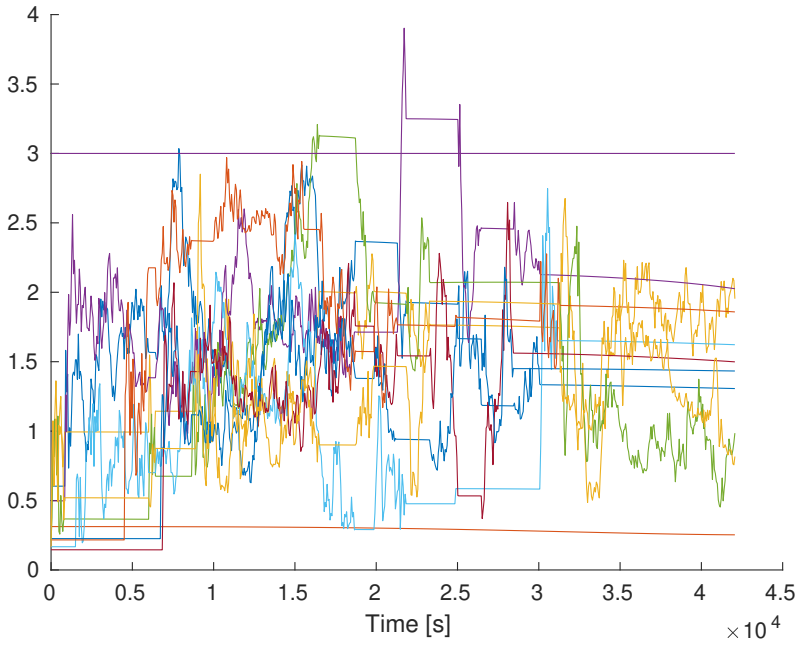

Figure 4.17: Mahalanobis distance of the full state for the Q-learning scenario.

The filter is predominantly consistent, but there are certain targets that do not recieve any measurements throughout the entire night. A more agressive exploration scheme with many more interactions with the environment is required to effectively and properly train the networks through Q-learning.

4.2 Multi-agent Scenario

Multi-agent learning is exceedingly difficult when the agents are closely interacting and updating the environment for every other agent. This scenario is a simple case to see if the PGQL or Q-learning algorithms can be utilized for the sensor tasking problem.

Rather then checking the filter consistency, the goal of this scenario is to determine how well the sensors are switching between the targets. Given the covariance ellipsoids in the two dimensional plot in Fig 3.3, the sensor on the x-axis should be focusing on the horizontally and diagonally moving target, and the sensor on the y-axis should be focusing on the vertically and diagonally moving target.

The actions for each sensor in both the off-policy Q-learning and the on-policy PGQL case can be seen in Tbls. 4.1 and 4.2.

Table 4.1: Actions for the multi-action Q-learning scenario over time.

Time [s]	Action for Sensor 1	Action for Sensor 2
1	2	1
2	2	1
3	2	0
4	2	0
5	2	1
6	2	0
7	2	0
8	2	0
9	2	0

Table 4.2: Actions for the multi-action PGQL scenario over time.

Time [s]	Action for Sensor 1	Action for Sensor 2
1	0	1
2	0	1
3	0	1
4	0	1
5	0	1
6	0	1
7	0	1
8	0	1
9	0	1

The actions in Tbls. 4.1 and 4.2 were after 18,000 interactions with the environment and were properly converged. The hidden layers of the neural networks are still set to three, but the size of each neuron is scaled by 10 from the design in the geosynchronous orbit scenario. Unfortunately, neither scenario seems to have worked well, but the second agent in the Q-learning scenario does vary between two different actions over time. The first agent, however, stays rigid on focusing on target 2.

Only a local reward was prescribed for each agent, i.e., the agent only recieved rewards for the observation that it found, instead of a portion of a global reward calculated using the state after both sensing actions were accomplished. A possible reason for the rigidness of the policies could be attributed to the lack of a global reward signal into the training. The agents, instead of coordinating with one another, attempt to specialize and largely stay independent of one another to maximize their individual rewards.

While the lack of a global reward played a part, another factor in the rigidity can be attributed to the number of interactions with the environment. With a longer training window, there is some promise that the Q-learning could slowly adapt to have the sensor look at many targets. The PGQL algorithm converged halfway through the training cycle and refused to utilize the Q-learning portion of the training step to shift its policies.

Chapter 5

Conclusions

This work presents an approach to solve the heterogenous, non-myopic sensor tasking problem utilizing mutli-target filters and reinforcement learning. To provide the state information in an acceptable formation for the reinforcement learning algorithms, the LMB fiter is utilized to provide an estimate of both the multi-target state pdf and the individual target pdfs. The LMB filter is derived from RFS theory, and while the filter utilized in this work is not Bayes optimal, a Bayes optimal filter can be derived from RFS theory. The reinforcement learning algorithms utilized in this work are new techniques developed in the last 5 years that are viable due to the computational capabilites of modern computers. The two techniques tested were the off-policy Q-learning and the mostly on-policy PGQL algorithm.

The motivation of this work was to test performance of these algorithms in the space situational awareness domain and so the main test was a single sensor case tracking and monitoring for space objects. This case was not extended to the multi-sensor case, and will have to be in the future. However, a simpler multi-agent case was tested to determine feasibility of both the on-policy and off-policy algorithms.

5.1 Observations

The Rényi divergence for the space situational awareness problem is better than the Cauchy-Schwarz divergence since the observations for a full space object catalog can be extremely sparse and the space objects are never fully contained in a single sensor field of regard. However, in this work, the Cauchy-Schwarz divergence provides an alternative to the Rényi divergence for quicker training. The Rényi divergence is numerically more stable. The Rényi divergence ranges for this work was limited to around 4, while the Cauchy-Schwarz divergence ranged from 1e14 to 1e16.

There are many tuning parameters in the neural network design and the reinforcement learning algorithms that can be changed to optimize the training, but that optimization was not performed. The training of both the PGQL algorithm and the Q-learning algorithm could be optimized for both performance and training time, but this work shows that even with potentially sub-optimal parameters, a solution can be found.

O'Donoghue, et al. [45], showed, overall, that the algorithm outperforms other reinforcement learning algorithms by receiving higher rewards from the environment. The PGQL algorithm vastly outperforms the Q-learning algorithm when compared using the same number of interactions with the environment. However, in terms of robustness a more off-policy method than PGQL with the same upgrades in training efficiency might be the best path forward.

For the simple multi-agent learning scenario, it seems that an off-policy

method might work the best. Since, exploration can be tuned directly rather than updating the cost function for the PGQL algorithm, a more robust network can be trained. However, shaping the reward by punishing independence and inducing more cooperation is the next step to fully explore the solution provided.

Chapter 6

Future Work

While reinforcement and machine learning provide an interesting solution to the sensor tasking problem, effort is still required to properly deploy these techniques for use in real applications. The goal of this chapter is to discuss the shortcomings of the techniques utilized here and, for some, discuss potential solutions.

6.1 Convolutional Neural Networks

Neural networks are constructed with a constant, known number of inputs which causes issues when a space catalog is to be constructed with a varying number of objects. The feedforward networks used in this work were constructed with the known information that the catalog would have three or 10 targets. Without this information, or with a varying number of objects it would be impossible to use a single feedforward network.

The sensors utilized to track space objects and debris are tracked on the ground with electro-optical or radar sensors. The data that these sensors provide can be constructed as a matrix of intensities representing the probabilities distribution in observations space. For right ascension and declination

measurements, an image-like object can be generated which convolutional networks excel at classifying. For a higher dimensional observation vector requires a corresponding higher dimensional "image". The number of inputs, especially for the test cases considered in this work, are significantly higher for convolutional networks, but are kept constant no matter the number of potential space objects.

The convolutional network also solves the problem of the large number of input neurons required for a moderately sized catalog. Each object translates to 12 inputs for a feedforward network, which for a moderately sized catalog can lead to long training times. Convolutional nets can be trained based on the field of regard of the sensor, which provides a fixed number of inputs for the network.

The neural network constructed as a convolutional network could also give a discretized field of view from the field of regard. While this should be possible with the feedforward networks used in this work, the convolutional networks are more intuitive when the output is a specific field of view. With each convolution and pooling, specific features are extracted which correspond to a specific field of view.

The inputs to the convolutional network also have an intuitive connection to the Rényi and Cauchy-Schwarz divergence. Instead of selecting the means of each target and the diagonal of the covariances as is the case in the feedforward network design in this work, the LMB state can be converted to the PHD state and those intensities can be mapped to an "image". The use

of a convolution network in this fashion would give a direct intution between input, reward, and output and is also beneficial to the training process.

6.2 Missed Detections and Operation Considerations

For an on-policy reinforcement learning algorithm, it is required that the actions taken have an appropriate reward correspond to it. However, missed detections or operators performing actions to a different schedule may destabilize the training of an on-policy network by not providing the appropriate reward. If missed detections or operator deviation were systematic, the neural network would be able to, ideally, learn this behavior. However, missed detections are modeled as a stochastic process and operator deviation can occur for a variety of reasons. Due to this, missed detections or operator deviation may cause instability during the training process.

For operation deviation, off-policy learning techniques such as Q-learning or off-policy policy gradient are able to train with data that does not correspond with the action generated by the neural network policy. These techniques still require that the action taken is known and can be stored for future training. For missed detections, if the entire space of possibilities is visited, i.e., there are enough samples that the learning algorithm is traversed, then missed detections may not be a problem for any reinforcement learning training scheme.

6.3 Reinforcement Learning Fragility

A properly tuned neural network with reinforcement learning can give powerful results, but at the cost of significant time in tuning the neural network. Each algorithm presented in this work is subject to these tuning parameters and can effect training by destabilizing the network or extensive training times. Additionally, each time a neural network is trained with the same data, it is not guaranteed that the neural network will converge on a similar policy, since reinforcement learning requires an infinite horizon to search over the entire scope of possibilities. The reinforcement and machine learning communities are attempting to develop algorithms such that the tuning parameters and length of training are not variables that affect the convergence rate and to ensure that each neural network that is trained are similar in performance [11].

As mentioned earlier in Chapter 2, multi-agent learning has not yet reached a state where it is robust or has any guarantees of performance. Most uses of multi-agent learning are ad hoc and a policy can be generated, but its optimality or general use is severely limited. Additionally, most of the multi-agent learning research currently is confined to advancing DQN methodologies to the multi-agent domain. DQN is preferred over policy gradients since it is more robust to changes in the environment caused by other agents [8, 13, 28]. The PGQL algorithm is predominantly on-policy and thus is subject to similar issues as vanilla policy gradient methods.

The sensor tasking problem can be redefined as a co-operative game between multiple sensors and stochastic game theory can be utilized to stabilize

the training process for each sensor agent. Q-learning can be stabilized by incorporating it with Nash equilibria to enforce some level of co-operation between the various agents [13, 46]. Nash equilibria is the policy that provides the optimal strategy for a singular agent given the other agents actions are optimal and fixed as well. An alternative to stochastic game theory approaches is to infer a model of the other agents policies via Monte Carlo samples and Bayesian inference, which also utilizes DQN [8]. DQN trains much slower in comparison to other algorithms, and so perhaps another off-policy analog to DQN can be utilized.

Appendices

Appendix A

Initial States for the Geosynchronous Satellites

Table A.1: Initial Cartesian states in kilometers and kilometers per second, as appropriate.

Target #	Initial State
1	$\begin{bmatrix}29248.452 & 30371.113 & 3.506 & -2.214 & 2.133 & -0.0001955\end{bmatrix}$
2	$\begin{bmatrix}28989.206 & 30632.488 & 6.996 & -2.232 & 2.113 & -0.000392\end{bmatrix}$
3	$\begin{bmatrix}29219.609 & 30394.770 & 472.505 & -2.2168 & 2.130 & -0.0256\end{bmatrix}$
4	$\begin{bmatrix}29274.224 & 30332.242 & 58.635 & -2.2119 & 2.136 & -0.0032\end{bmatrix}$
5	$\begin{bmatrix}29144.468 & 30482.801 & 349.783 & -2.2212 & 2.125 & -0.0196\end{bmatrix}$
6	$\begin{bmatrix}29179.925 & 30450.287 & -175.513 & -2.2189 & 2.127 & 0.0097\end{bmatrix}$
7	$\begin{bmatrix}29299.014 & 30318.090 & -474.337 & -2.2137 & 2.133 & 0.0254\end{bmatrix}$
8	$\begin{bmatrix}29246.121 & 30369.256 & -472.812 & -2.2150 & 2.132 & 0.0255\end{bmatrix}$
9	$\begin{bmatrix}29095.326 & 30531.169 & -232.896 & -2.2242 & 2.122 & 0.0131\end{bmatrix}$
10	$\begin{bmatrix}29299.014 & 30318.090 & 474.337 & -2.2137 & 2.133 & -0.0254\end{bmatrix}$

Bibliography

[1] Martín Abadi and et. al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] M. Beard, B. T. Vo, B. N. Vo, and S. Arulampalam. Sensor control for multi-target tracking using cauchy-schwarz divergence. In *2015 18th International Conference on Information Fusion (Fusion)*, pages 937–944, July 2015.

[3] Dimitri P. Bertsekas and David A. Castanon. Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics*, 5(1):89–108, Apr 1999.

[4] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1st edition, 1996.

[5] K. J. DeMars, I. I. Hussein, M. K. Jah, and R. S. Erwin. The cauchy-schwarz divergence for assessing situational information gain. In *2012 15th International Conference on Information Fusion*, pages 1126–1133, July 2012.

[6] R. Van der Merwe and E. A. Wan. The square-root unscented kalman filter for state and parameter-estimation. In *2001 IEEE International Con-*

ference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221), volume 6, pages 3461–3464 vol.6, 2001.

[7] R. S. Erwin, P. Albuquerque, S. K. Jayaweera, and I. Hussein. Dynamic sensor tasking for space situational awareness. In *Proceedings of the 2010 American Control Conference*, pages 1153–1158, June 2010.

[8] Jakob N. Foerster, Nantas Nardelli, Gregory Farquhar, Philip H. S. Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. *CoRR*, abs/1702.08887, 2017.

[9] Steven Gehly, Brandon Jones, and Penina Axelrad. Sensor allocation for tracking geosynchronous space objects. *Journal of Guidance, Control, and Dynamics*, 41(1):149–163, 2018/03/02 2016.

[10] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *ArXiv e-prints*, June 2014.

[11] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *ArXiv e-prints*, January 2018.

[12] Matthew J. Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR*, abs/1507.06527, 2015.

[13] Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *CoRR*, abs/1603.01121, 2016.

[14] Alfred O. Hero, Bing Ma, Olivier Michel, and John Gorman. Alpha-divergence for classification, indexing and retrieval. Technical report, UNIVERSITY OF MICHIGAN, 2001.

[15] Alfred Olivier Hero, David Castan, Doug Cochran, and Keith Kastella. *Foundations and Applications of Sensor Management*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[16] Hung Gia Hoang, Ba-Ngu Vo, Ba-Tuong Vo, and Ronald P. S. Mahler. The cauchy-schwarz divergence for poisson point processes. *CoRR*, abs/1312.6224, 2013.

[17] B. A. Jones. Cphd filter birth modeling using the probabilistic admissible region. *IEEE Transactions on Aerospace and Electronic Systems*, 54(3):1456–1469, June 2018.

[18] K. Kastella. Discrimination gain to optimize detection and classification. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 27(1):112–116, Jan 1997.

[19] TS Kelso. Analysis of the 2007 chinese asat test and the impact of its debris on the space environment. In *8th Advanced Maui Optical and Space Surveillance Technologies Conference, Maui, HI*, volume 7, 2007.

[20] TS Kelso et al. Analysis of the iridium 33-cosmos 2251 collision. *Advances in the Astronautical Sciences*, 135(2):1099–1112, 2009.

[21] Chris Kreucher, Doron Blatt, Alfred Hero, and Keith Kastella. Adaptive multi-modality sensor scheduling for detection and tracking of smart targets. *Digital Signal Processing*, 16(5):546 – 567, 2006.

[22] Chris Kreucher, Keith Kastella, and Alfred O. Hero III. Sensor management using an active sensing approach. *Signal Processing*, 85(3):607 – 624, 2005.

[23] V. Krishnamurthy and R. J. Evans. Hidden markov model multiarm bandits: a methodology for beam scheduling in multitarget tracking. *IEEE Transactions on Signal Processing*, 49(12):2893–2908, Dec 2001.

[24] Guillaume Lample and Devendra Singh Chaplot. Playing FPS games with deep reinforcement learning. *CoRR*, abs/1609.05521, 2016.

[25] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[26] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436 EP –, 05 2015.

[27] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus Robert Müller. *Efficient BackProp*, pages 9–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[28] Joel Z. Leibo, Vinícius Flores Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. *CoRR*, abs/1702.03037, 2017.

[29] Y. Li, L.W. Krakow, E.K.P. Chong, and K.N. Groom. Approximate stochastic dynamic programming for sensor scheduling to track multiple targets. *Digital Signal Processing*, 19(6):978 – 989, 2009. DASP'06 - Defense Applications of Signal Processing.

[30] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

[31] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3):293–321, May 1992.

[32] R. Linares and R. Furfaro. Dynamic Sensor Tasking for Space Situational Awareness via Reinforcement Learning. In *Advanced Maui Optical and Space Surveillance Technologies Conference*, page 36, September 2016.

[33] R. Linares and R. Furfaro. An Autonomous Sensor Tasking Approach for Large Scale Space Object Cataloging. In *Advanced Maui Optical and Space Surveillance (AMOS) Technologies Conference*, page 55, 2017.

[34] Keqin Liu and Qing Zhao. Indexability of restless bandit problems and optimality of whittle's index for dynamic multichannel access. 10 2008.

[35] Ronald P. S. Mahler. *Statistical Multisource-Multitarget Information Fusion*. Artech House, Inc., Norwood, MA, USA, 2007.

[36] James Manyika and Hugh Durrant-Whyte. *Data Fusion and Sensor Management: A Decentralized Information-Theoretic Approach*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1995.

[37] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[38] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[39] Katta G. Murty. An algorithm for ranking all the assignments in order of increasing cost. *Operations Research*, 16(3):682–687, 1968.

[40] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Trust-pcl: An off-policy trust region method for continuous control. *CoRR*, abs/1707.01891, 2017.

[41] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.

[42] Sean M. McArdle Brandon A. Jones Nicholas Ravago, Akhil K. Shah. Large constellation tracking using a labeled multi-bernoulli filter. volume 10646, pages 10646 – 10646 – 20, 2018.

[43] M. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[44] J. Le Ny, M. Dahleh, and E. Feron. Multi-uav dynamic routing with partial observations using restless bandit allocation indices. In *2008 American Control Conference*, pages 4220–4225, June 2008.

[45] Brendan O'Donoghue, Rémi Munos, Koray Kavukcuoglu, and Volodymyr Mnih. PGQ: combining policy gradient and q-learning. *CoRR*, abs/1611.01626, 2016.

[46] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, November 2005.

[47] Julien Pérolat, Joel Z. Leibo, Vinícius Flores Zambaldi, Charles Beattie, Karl Tuyls, and Thore Graepel. A multi-agent reinforcement learning model of common-pool resource appropriation. *CoRR*, abs/1707.06600, 2017.

[48] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682 – 697, 2008. Robotics and Neuroscience.

[49] Warren B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality (Wiley Series in Probability and Statistics)*. Wiley-Interscience, 2007.

[50] S. Reuter, B. T. Vo, B. N. Vo, and K. Dietmayer. The labeled multi-bernoulli filter. *IEEE Transactions on Signal Processing*, 62(12):3246–3260, June 2014.

[51] B. Ristic, B. N. Vo, and D. Clark. A note on the reward function for phd filters with sensor control. *IEEE Transactions on Aerospace and Electronic Systems*, 47(2):1521–1529, April 2011.

[52] Simo Särkkä. *Bayesian Filtering and Smoothing*. Cambridge University Press, New York, NY, USA, 2013.

[53] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015.

[54] D. Schuhmacher, B. T. Vo, and B. N. Vo. A consistent metric for performance evaluation of multi-object filters. *IEEE Transactions on Signal Processing*, 56(8):3447–3457, Aug 2008.

[55] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[56] Jeffrey K. Uhlmann Simon J. Julier. New extension of the kalman filter to nonlinear systems. volume 3068, pages 3068 – 3068 – 12, 1997.

[57] Z. Sunberg, S. Chakravorty, and R. Erwin. Information space sensor tasking for space situational awareness. In *2014 American Control Conference*, pages 79–84, June 2014.

[58] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, Aug 1988.

[59] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

[60] David A. Vallado. *Fundamentals of Astrodynamics and Applications*. McGraw-Hill, 1997.

[61] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[62] B. T. Vo and B. N. Vo. Labeled random finite sets and multi-object conjugate priors. *IEEE Transactions on Signal Processing*, 61(13):3460–3475, July 2013.

[63] K. Wang. Optimality of myopic policy for restless multiarmed bandit with imperfect observation. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, Dec 2016.

[64] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015.

[65] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[66] Patrick S. Williams, David B. Spencer, and Richard S. Erwin. Coupling of estimation and sensor tasking applied to satellite tracking. *Journal of Guidance, Control, and Dynamics*, 36(4):993–1007, 2018/03/02 2013.

[67] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992.